CDRL: B003
29 January 1994

UNISYS

Training Plan
Central Archive for Reusable Defense Software
(CARDS)

Informal Technical Report

Central Archive for Reusable Defense Software

STARS-VC-B003/001/00
29 January 1994

CDRL: B003
29 January 1994

INFORMAL TECHNICAL REPORT For The SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS)

Training Plan
Central Archive for Reusable Defense Software
(CARDS)

STARS-VC-B003/001/00
29 January 1994

Contract NO. F19628-93-C-0130
Line Item 0002AB

Prepared for:

Electronic Systems Center
Air Force Material Command, USAF
Hanscom AFB, MA 01731-2816

Prepared by:

Azimuth, Inc.
and
Electronic Warfare Associates, Inc.
under contract to
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

Distribution Statement "A"
per Dod Directive 5230.24
Approved for public release, distribution is unlimited

CDRL: B003
29 January 1994

INFORMAL TECHNICAL REPORT For The SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS)

Training Plan
Central Archive for Reusable Defense Software
(CARDS)

STARS-VC-B003/001/00
29 January 1994

Contract NO. F19628-93-C-0130
Line Item 0002AB

Prepared for:

Electronic Systems Center
Air Force Material Command, USAF
Hanscom AFB, MA 01731-2816

Prepared by:

Azimuth, Inc.
and
Electronic Warfare Associates, Inc.
under contract to
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

CDRL: B003
29 January 1994

Data ID: STARS-VC-B003/001/00

CDRL: B003
29 January 1994

INFORMAL TECHNICAL REPORT
Training Plan
Central Archive for Reusable Defense Software
(CARDS)

Principal Author(s):

Kerrin Smith *Date*

Harry J. Facemire *Date*

Approvals:

System Architect: *Kurt Wallnau* *Date*

Program Manager: *Lorraine Martin* *Date*

(Signatures on File)

PREFACE

This document is an update of the original Training Plan. The original version was co-authored by Kerrin Smith and Leslie Hayhurst. The documentation and courses developed and presented by the CARDS team are updated periodically as new input is received and new developments in technology occur. These changes and additions are reflected in this updated version of the Training Plan.

Due to the similarity and overlapping of course content, the training course for DoD contractors and the training course for DoD organizations have been combined into a single course entitled *Management Level Training Course.* Appendix A is now an outline of a course entitled *Application Engineering with Domain-Specific Reuse.* Appendix B is an outline of the *Introduction to Reuse* course which was developed and presented by the CARDS team. As new courses are developed and delivered to the Government, they will be added as appendices.

Appendices A, B, and C from the previous version have been incorporated into this version as handouts in the present Appendix A and B. Other changes include minor style changes as well as updates of the references and reading lists.

ABSTRACT

The Central Archive for Reusable Defense Software (CARDS) Training Plan serves as a comprehensive guide for creating training courses and training materials on domain-specific reuse for Department of Defense (DoD) organizations, DoD contractors, system engineers, and university professors. The Training Plan provides guidance for conducting three different domain-specific software reuse training courses for four different audiences. The intent of the courses in the Training Plan is to demonstrate how software reuse can reduce development and maintenance time and costs, reduce project risks, and increase productivity.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Department of the Air Force ESC/ENS Hanscom AFB, MA 01731-2816	10. SPONSORING/MONITORING AGENCY REPORT NUMBER B003
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION AVAILABILITY STATEMENT DISTRIBUTION "A"	12 b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Central Archive for Reusable Defense Software (CARDS) Training Plan serves as a comprehensive guide for creating training courses and training materials on domain-specific reuse for Department of Defense (DoD) organizations, DoD contractors, system engineers, and university professors. The Training Plan provides guidance for conducting three different domain-specific software reuse training courses for four different audiences. The intent of the courses in the Training Plan is to demonstrate how software reuse can reduce development and maintenance time and costs, reduce project risks, and increase productivity.

Table of Contents

1 INTRODUCTION

1.1 PURPOSE

This Training Plan serves as a comprehensive guide for creating training courses and training materials on domain-specific reuse for Department of Defense (DoD) organizations, DoD contractors, system engineers, and university professors.

1.2 RATIONALE

A foundation for integrating software reuse into the business practices of DoD organizations and contractors is necessary if the full potential of software reuse is to be realized. Education and training is the foundation upon which new concepts are incorporated into existing processes. Training and education guidelines must be in place to support the inclusion of reuse in the software development process.

1.3 SCOPE

The Training Plan is a recommendation of how top quality domain-specific software reuse training should be conducted. The Plan identifies the generic functions necessary to support these recommendations. For instance, the Plan itself is detached from dollar figure amounts, but a Price Worksheet (see Section 5.6) is provided from which these amounts may be derived. Implementing individual courses will require omitting, adding, and/or modifying the plan's recommendations and outlined course content for different audiences. These changes will ultimately produce a tailored implementation of the plan each time a course is presented to an audience.

The Training Plan provides guidance for conducting three different domain-specific software reuse training courses for four different audiences. One of the courses, the Management Level Training Course, is directed at both DoD and contractor management level personnel. This course is intended to provide an understanding of the management infrastructure required to support software reuse. The second course, the Training Course for System and Software Engineers, has been developed to provide system/software engineers with an understanding of how to incorporate software reuse into their current system/software development processes. The third course, a Training Course for Universities, provides outlines on how to incorporate software reuse into university curricula.

The intent of the courses in the Training Plan is to demonstrate how software reuse can reduce development and maintenance time and costs over time, reduce project risks, and increase productivity.

1.4 DEVELOPMENT OF THE TRAINING PLAN

1.4.1 Approach

The purpose of the Training Plan is to specify the approaches to be used in training DoD contractors, university students and professors, and DoD organization personnel in domain-specific reuse. The Training Plan contains outlines (identifying content, methods, and duration) of curriculum contents for each of the target audiences. This approach allows for a wide audience characterization, and contains a tremendous amount of information. From these course outlines contained in the Training Plan, CARDS used a format from DISA to prepare course materials to enhance the exchangeability of the courses among DoD efforts.

1.4.2 Identification of Courses

In developing the three course outlines, careful consideration was given to the composition of the intended audiences, and how domain-specific reuse impacts their respective functions. The four audiences identified are: management in DoD organizations, management in DoD contractors, system/software engineers, and university professors and administrators.

The common business processes that establish a working relationship between DoD organizations and DoD contractors were identified to assist in depicting the specific content areas for each of the DoD courses. These processes include developing acquisition plans, requests for proposals (RFPs), statements of work (SOWs), and proposals.

Introducing new technology, such as reuse, may require both managerial and implementation changes throughout all of these business processes. The DoD Organization is identified as initiating the acquisition plans and the RFPs in this working relationship. The DoD Contractor is identified as responding to the DoD Organization's RFP with an appropriate proposal.

The course for DoD organizations and DoD contractors is structured to provide management level personnel with an understanding of reuse and to characterize potential solutions for integrating domain-specific reuse into their particular business practices. The key issue in the modification of the course for these organizations is the focus on their respective views of the business processes and what their individual responsibilities are in making reuse work.

To further refine the content of each course, a distinction was made between managerial related issues and their actual implementation at the technical level. The System/Software Engineer's course was organized to address the technical implementation of reuse. This course is independent of DoD Organization and DoD Contractor distinguishing characteristics at the technical level. Consequently, the course is designed to be applicable to both DoD organizations and DoD contractors.

The System/Software Engineer's course objective is to provide a basis for integrating domain-specific reuse into current system/software engineering practices and to explore how the system/software engineer can use domain analysis products.

The third course addressed by this Plan is targeted toward university professors. The main objective of this course is to provide an understanding of software reuse and to identify potential avenues for integrating domain-specific reuse into their courses and curriculum.

1.4.3 Bibliography for Developing the Training Plan

1. Bacon, T. R., and Frierman, L. H. Shipley Associates Style Guide, 1990.
2. Bugelski, B. R. The Psychology of Learning Applied to Teaching. Bobbs-Merrill Company, 1964.
3. Computer Maintenance Training Manual. NSA, M-5099, 1 Oct 1967.
4. Computer Program Training Plan. Navy-AS, UDI-H-21262, AIR-533, 7 Sep 1973.
5. Dwyer, F. M. Strategies for Improving Visual Learning. Learning Services, 1978.
6. How to Prepare and Conduct Military Training. FM 21-6. HQ, Department of the Army, Nov 1975.
7. Proposed Curricula for Librarians Course. Asset Source for Software Engineering Technology (ASSET), National Software Technology Repository.
8. STARS Reuse Concept of Operation. Task US30: Vol. I Ver. 0.5 - DRAFT STARS-SC-03725/001/00, 27 Aug 1991.
9. Steinger, L. The Career Development Program for Acquisition Personnel. DoD Report 5000.52-M, 15 Nov 1991.
10. Student and Training Course Evaluation Forms. CDRL Item A028, Contract F19630-88-D-0004.
11. Student's Training Course Guide. DoD, DI-H-7071, NAVAIE 04B1, 8 Feb 1981.
12. Student Training Materials. NTEC, DI-H-25724B, NAVTRAEQUIPCEN Code N-25, 30 Jun 1986.
13. Tomayko, J. E. Software Configuration Management. SEI Curriculum Module SEI-CM-4-1.3 (Preliminary), Jul 1987.
14. Training and Equipment Plan. DoD, DI-H-7066, NAVAIR 041B, 18 Feb 1981.
15. Training Development and Support Plan. F/AFSPACECOM-1013 CCTS, DI-MGMT-80476, 26 Aug 1987.

16. Warriner's English Grammar and Composition, Fifth Course. New York: Harcourt Brace Jovanovich, 1982.

1.4.4 Individual Course Information

For each of the three training courses, the Plan includes a section on course characteristics, an outline of the course content, recommended instructor qualifications, and references used to develop the course.

The sections on course characteristics provide a course overview (including a rationale for conducting the training), the course objectives, the course's completion criteria, an audience characterization, and a student reading list for their respective courses.

The instructor should use the student reading list to develop suggested prerequisite readings, course texts, and supplemental readings. The supplemental reading list should provide students with a collection of documents that will be useful in advancing their understanding of domain-specific reuse beyond the information covered in the course.

2 MANAGEMENT LEVEL TRAINING COURSE

Reuse of software system components is being promoted as supporting a focus toward more cost-effective and timely development of software-intensive systems. Domain-specific reuse provides a framework and method to effectively reuse common domain components. The purpose of this course is to provide management-level DoD organizations and contractor personnel with an understanding of the benefits of reuse and what is required to support the integration of domain-specific reuse into the management processes.

2.1 COURSE CHARACTERISTICS

2.1.1 Rationale

Reuse is a long term investment and without ongoing support and incentives, reuse programs will not be successful. Inadequate training and materials exist for director-level, mid-level, and team leader personnel in Government, industry, and academia in the areas of how and why to implement software reuse, and how to address perceived and real barriers to effective software reuse.

2.1.2 Course Objectives

The learning objective for this course is to provide management level personnel with:

- an understanding of the benefits of reuse;
- an understanding of the background of reuse and domain-specific reuse;
- an understanding of the responsibilities involved in providing a reuse infrastructure;
- an understanding of how reuse may impact writing acquisition plans, RFPs and SOWs, and evaluating submitted proposals;
- the ability to respond to RFPs with proposals that incorporate reuse;
- an understanding of the methods for integrating domain-specific reuse into current business practices;
- an understanding of how to integrate reuse.

2.1.3 Audience Characterization

One of CARDS' many goals is to identify and outline the education and training necessary for each audience to make widespread reuse a reality. To do this, CARDS must determine our audience and the level of their capacity towards reuse utilization and/or implementation. CARDS defines four levels of functionality:

Level I. Vice Presidents

Direction Level Managers

Program Executive Officers

Designated Acquisition Commanders

University Department Heads

Level II. Mid-level Managers

Program Manager

University Professors

Level III. Unit Managers

Project Leaders

Team Leaders

University Instructors

Level IV. Individual Contributors

Implementation Engineers

University Students

The intended audience for this training course is personnel at level 1 and level 2. The primary responsibility of these personnel is strategic management of their business, especially direction-level managers and program managers.

Direction-level managers are those responsible for implementing cross-program infrastructure change and instituting policy change in marketing practices and/or system development methods. They are also responsible for instantiating top-level policy guidance and change, managing program costs, and instituting policy change in procurement practices and/or system development methods. Direction-level managers also possess interests in domain-specific reuse, management level awareness of technology, cost and benefit issues, and long term benefits. They also possess an awareness of their application domain.

Program managers are responsible for planning and executing acquisition programs, and instantiating program level policy guidance/change. They are also responsible for risk assessment, analysis, and mitigation. Program managers possess an interest in domain-specific reuse, knowledge of basic system acquisition, knowledge about general software development issues and management level awareness of technology. They also possess an awareness of cost, scheduling, and requirements.

2.1.4 Course Completion Criteria

Participants will apply concepts and techniques identified by the course's content areas, producing products as proof of completing the training course. Upon completion of the training course the participants will be able to:

- identify areas within their organizations that should support reuse;
- outline specific procedures for implementing reuse;
- develop plans for identifying applicable domain areas and reuse infrastructures within their business area;
- incorporate reuse concepts into RFPs and SOWs;
- define an appropriate organizational structure to support reuse;
- identify strategies for incorporating reuse into proposals;
- develop reuse-oriented evaluation criteria in relation to RFPs and SOWs.

Upon completion of the Training Course, the participants will have been instilled with certain attributes and abilities which are listed with their respective levels.

Level I. Vice Presidents:

Commitment to Reuse,

Knowledge of Reuse,

Allocate resources to establish Reuse,

Provide vision and strategies concerning:

Environment framework,

Metrics, and

Incentives,

Continued Involvement.

Level II. Mid-level Managers:

Commitment to Reuse,

Knowledge of Reuse,

Identification and allocation of resources,

Implementation plan for vision and strategies concerning:

- Environmental Framework,
- Personnel,
- Tools,
- Processes,
- Metrics, and
- Incentives,

Continuing Involvement, and

Management boundaries-interface with external resources.

Level III. Unit Managers:

Implement Plan in the form of:

- Processes (collect/analyze),
- Metrics, and
- Incentives.

Commitment to Reuse,

Employ resources,

Continued Involvement,

Reusing,

Management of boundaries - interface with external resources, and

Lessons Learned.

Level IV Individual Contributors:

Commitment to Reuse,

Knowledge of Reuse,

Rationale for Reuse,

Continuous Improvement,

Reusing, and

Lessons Learned.

2.1.5 Student Readings

1. Acquisition Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04105/001/00, 30 Apr 93.
2. Arnold, R. S., Frakes, W., and Prieto-Diaz, R. Software Reuse, Domain Analysis, and Reengineering. Conference Notes, 6-8 Apr 1992.
3. Direction Level Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04104/001/00, 20 Nov 92.
4. STARS Reuse Concepts, Volume 1, Conceptual Framework for Reuse Processes (CFRP), Version 2, STARS-UC-05159/001/00, 13 Nov 92.
5. Williams, A. SWAN-An Ada Program for Cost Estimation. AdaIC Newsletter, Sep 1991.
6. Wong, W. A Management Overview of Software Reuse. NBS Special Publication 500-142, Washington, DC: USGPO, Sep 1983.

2.2 OUTLINE OF RECOMMENDED COURSE CONTENT

The following is an outline of the recommend course content.

2.2.1 Reuse as Part of the Solution

A. The Dilemma:

1. Software industry;

   a. Increasing software costs,

   b. Increasing complexity of software,

c. Increasing development time,

d. Shortage of experienced personnel,

e. Degradation of quality,

f. Software as a commodity,

g. Software as a utility.

2. Customer;

a. Demand for high quality products,

b. Demand for low risk products,

c. Demand for automated programming support,

d. Demand for customized software packages,

e. Demand for flexible products,

f. Demand for faster product delivery.

B. A Solution:

1. What is Reuse?

a. The process of incorporating into the life-cycle of a software system any preexisting components (e.g., requirements, design, code, executables).

b. Examples.

2. Why Reuse?

a. Reduces schedule,

b. Increases productivity;

Improves programming capabilities,

Reduces the amount of documentation and testing.

c. Increases quality;

Software is well designed (for reuse),

Software is well documented (standard),

Software is well tested (certified),

Software is well understood (functionality),

Concepts are manifested in rapid prototyping.

d. Increases reliability,

e. Reduces maintenance costs,

f. Overall, reuse allows for more system development within the same time frame,

g. Reduces long term development costs,

h. Increases flexibility,

i. Improves competitive posturing.

3. Evolution of Reuse;

a. Early reuse projects,

b. Reuse success stories,

c. Current approaches and concepts:

Components and wide spectrum reuse,

Composition versus generation,

Design for reuse,

Design with reuse,

Life-cycle importance of reuse,

Architecture driven,

Domain-specific reuse.

d. Reuse is NOT:

A cure all,

Object oriented design,

A specific technology.

2.2.2 Introduction to Domain-Specific Reuse

A. What is a Domain?

1. A domain is a set of common capabilities and data constituting a set of current and future systems in a particular application area, such as:

   a. A business area,

   b. A software business area,

   c. A software intensive application area,

   d. An application area for which similar software systems have been built.

2. Examples.

B. What is Domain-Specific Reuse?

1. Reuse of ideas, knowledge, artifacts, personnel, and components in an existing domain.

2. Examples.

C. What is Architecture-Centric Reuse?

D. What is Process-Driven Reuse?

E. What is Library-Assisted Reuse?

F. Why Domain-Specific Reuse?

1. Reuse is more effective in a narrow, well defined domain where similar systems are built.

2. Examples.

G. Domain Selection Checklist.

1. Define strategy.
2. Select domain analysis resources.
3. Identify boundaries/scope of domain.
4. Select mature, stable, well-defined domain.
5. Define predictable technology.
6. Perform market evaluation.
7. Identify available domain expertise.
8. Conduct readiness cost and benefit analysis.

H. Existing Technological Support.

1. STARS efforts,
2. CARDS,
3. PRISM,
4. DISA (DSRS),
5. RAASP,
6. Others.

2.2.3 Considerations When Integrating Reuse

A. Strategic Planning:

1. Return on Investment Analysis:

   An organization must consider the costs and benefits of implementing reuse. Some of the questions that can be asked are:

   a. What investment does reuse require?

   b. Is a reuse program economically feasible?

c. What are the alternatives to a reuse program?

d. What alternatives exist for implementing a reuse program (technical support)?

e. What is the scope of the reuse program?

f. At what organizational level is the program targeted?

g. Some of the existing software cost models that incorporate reuse are:

SPC,

Reuse COCOMO,

IDA/STARS,

REVIC.

2. Feasibility Analysis:

An organization must decide if it is feasible to incorporate reuse into their current business practices. Some of the questions that can be asked to decide this strategic factor are:

a. How many similar systems will be built?

b. Is reuse beneficial?

c. Does the organization want to incorporate reuse?

d. Does the organization have the resources?

e. Is management committed?

f. Are implementation variations small or large?

g. Is existing software already available for reuse?

h. Is software production large enough to justify a reuse program?

i. Is a phased approach feasible?

j. Will the organization be more attractive to potential customers if reuse is incorporated?

k. Will the organization be more competitive if reuse is incorporated?

l. How well is the organization structured to support reuse?

m. What is the potential impact on business models?

3. Setting Goals and Objectives:

a. Establish a vision and strategy for achieving reuse for one organization.

b. Select a life-cycle process model for reuse based development.

c. Set specific reuse goals and objectives; define associated metrics.

d. Include both management and engineering processes, assume iterative planning, continuous process improvements.

4. Domain Suitability:

An organization must determine if its current operational domain is suitable for reuse. The following are some of the questions to help answer this issue:

a. Is the domain broad or narrow?

b. Is the domain mature and well understood?

c. Is the domain stable or changing continually?

d. Is the domain based on well established principles, methods, and formalisms?

5. Legal/Acquisition Issues:

a. Artifacts reused:

Domain model,

Software architecture,

Product design,

Implementation components.

b. Issues:

Ownership/Copyright/Proprietary issues,

Liabilities and responsibilities,

Contractual requirements,

Derivative works.

B. Implementation Planning:

1. Supports long-term commitment to reuse,

   a. Develop implementation plan.

2. Strong Management Involvement,

3. Characteristics of a Reuse Infrastructure:

   a. Stable: the same structure supports all stages of the same program.

   b. Flexible: the roles and people can be changed without affecting function.

   c. Evolving: reuse may start with a minimum set of one person in multiple roles and evolve into multiple teams with specific roles.

   d. Practical: a reuser can actually practice reuse.

   e. Effective: to the extent that all the elements of the infrastructure are efficient.

   f. Economical: cost, complexity, and sophistication are adjustable to available budget.

4. Develop Investment Strategy:

   a. Infrastructure Alternative Resources:

      Hire experienced reuse personnel,

      Hardware/Software purchases,

      Obtaining and testing components,

      Developing components.

   b. Education/training programs:

Management level,

Technical.

c. Setting up a reuse infrastructure:

Developing a domain model,

Developing an architectural model,

Developing a library model,

Building and maintaining a library.

d. Supporting the reuse infrastructure (library/technical) support.

5. Allow Time for Transition:

   a. Phased implementation,

   b. Iterative implementation.

6. Provide Continuous Financial Support:

   a. Resources,

   b. Education/training,

   c. Library/technical,

   d. Monitor reuse community,

   e. Long-term investment.

7. Establish, Monitor and Improve Incentive Programs:

   a. Individual,

   b. Corporate,

   c. Monetary,

   d. Other appropriate rewards.

8. Develop Accommodating Business and Original Equipment Manufacturer (OEM) Practices:

   a. Reusable Software Acquisition Factors:

      Identify effects on current acquisition and/or OEM policies,

      Incorporate reuse into existing acquisition and/or OEM policies,

      Introduce reusable software acquisition and/or OEM guide books and handbooks.

   b. Developing RFPs which incorporate Reuse:

      Locating and evaluating components including domain models and architectural models,

      Listing applicable Government reuse libraries,

      Listing COTS/GOTS products,

      Encourage continuous identification of additional library products,

      Identify status of software rights,

      Criteria for award based on reuse.

   c. Developing SOWs which incorporate Reuse:

      Management related tasks regarding subcontracting software,

      Impact life-cycle development,

      Document contract/legal issues,

      Identify reuse libraries,

      Define personnel requirements.

   d. Evaluating RFPs which specify reuse:

      Identifying applicable Government reuse libraries,

      Locating COTS products,

Identifying additional library products,

Identifying status of software rights,

Understanding criteria for award based on reuse.

e. Evaluating SOWs which require reuse:

Understand management related tasks regarding subcontracting software,

Impact on life-cycle development,

Evaluate contract/legal issues,

Identify reuse libraries,

Evaluate personnel requirements.

f. Developing Proposals incorporating reuse:

Requirements,

Specifications,

Additional expected costs,

Added benefits,

Risk reduction,

Indication of technical expertise in approach,

Proposal evaluation criteria.

g. Evaluating Proposals involving reuse:

Requirements,

Specifications,

Additional expected costs,

Added benefits,

Risk reduction,

Indication of technical expertise in approach,

Proposal evaluation criteria.

9. Establish a Reuse Measurement Plan:

   a. Management measurement,

   b. Technical measurement,

   c. Support incentives and objectives.

10. Management of Change:

   a. Social/cultural biases,

   b. Confusion with new technology,

   c. Loss of experienced personnel,

   d. Team effort required.

11. Sustaining Reuse in Your Organization:

   a. Incrementally staged,

   b. Formal programs,

   c. Systematic approach and evaluation,

C. Current technology issues.

2.2.4 Continuous Assessment and Improvement

A. Monitor external reuse issues:

1. Monitor and adjust reuse implementations,

2. Monitor external reuse communication and incorporate state of the art technology,

3. Monitor and adjust ongoing reuse activities, metrics, progress toward goals,

4. Pro-active systematic organization sensing,

5. Continuing advertisement to the organization/internal marketing:

   a. Allow awareness,

   b. Give feedback.

B. Lessons Learned:

1. Points Confirmed:

   a. Reuse program changes the software development process,

   b. Technology is important, but not essential,

   c. Reuse is more effective in narrow, well defined domains,

   d. Infrastructure support is essential,

   e. Classification is instrumental in domain understanding,

   f. Reuse is financially successful.

2. Factors Which Contribute to Failure:

   a. Lack of management involvement,

   b. No incentives,

   c. No procedures,

   d. Not enough information in library component catalog,

   e. Poor classification,

   f. No automated library,

   g. Original parts not designed for reuse.

2.2.5 Demonstration of Reuse Library

A. Connecting to the Library,

B. Log-in procedures,

C. Component selection and retrieval mechanisms,

D. Prototyping,

E. Log-out procedures.

2.3 TRAINING INSTRUCTOR

This section outlines the responsibilities and desired qualifications of the training instructor for the Management Level Training Course. These optimum qualifications are only recommendations; they describe the best candidate for this position. In the event that it is not possible to locate an individual who meets all of the detailed qualifications, some of the qualifications may be relaxed.

2.3.1 Job Description

The training instructor is responsible for conducting a tailored implementation of the recommended course content through a lecture/workshop format. This is accomplished by completing the following tasks:

- consulting the documents listed in Section 2.4, Bibliography, for developing course content;
- identifying specific implementor-related actions for reuse integration; updating and tailoring the recommended outline of course content with respect to management level personnel;
- preparing appropriate training materials;
- estimating the duration of the training session based on a blend of optimum cost-effectiveness and covering the course content thoroughly;
- conducting the training session;
- seeing that the atmosphere of the training session is informal and relaxing, intellectually stimulating, and learner-centered;
- integrating lecture, discussion, and small working groups into the training session;
- leading the course completion criteria workshop; and
- providing an evaluation of the course completion criteria.

2.3.2 Formal Education

The instructor should hold at least a BS degree in business administration (an MS degree is preferred) and a degree in computer science, system engineering, or a closely related field.

2.3.3 Knowledge of Instruction

The instructor should possess an understanding of the principles of learning, the methods of teaching, an ability to apply these principles and methods, and excellent communication skills.

2.3.4 Practical Teaching Experience

The instructor should possess teaching experience (at least two years) in planning short courses and seminars, and in addressing management level personnel.

2.3.5 Knowledge of Subject

The Instructor should possess:

- experience in the DoD's contractual and management processes;
- knowledge and an understanding of the impact of integrating reuse;
- experience in DoD procurement practices, including specific experience in writing RFPs and SOWs, and evaluating proposals;
- work experience at the DoD organization management level;
- practical experience in reuse library operations; and
- work experience in software development.

2.4 BIBLIOGRAPHY FOR DEVELOPING MANAGEMENT LEVEL TRAINING COURSE CONTENT

The training instructor should consult the following documents to prepare the tailored implementation of the course content and the training materials.

1. Acquisition Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04105/001/00, 30 Apr 93.

2. Arnold, R. S., Frakes, W., and Prieto-Diaz, R. Software Reuse, Domain Analysis, and Reengineering. Conference Notes, 6-8 Apr 1992.

3. Direction Level Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04104/001/00, 20 Nov 92.

4. Matsumoto, Y. Some Experience in Promoting Reusable Software: Presentation in Higher Abstract Levels. IEEE Transaction on Software Engineering, Vol. SE-10 No. 5, Sep 1984.

5. Tracz, W. Ada Reusability Efforts: A Survey of the State of the Practice. Proceedings of the Joint Ada Conference, Fifth National Conference on Ada Technology and Washington Ada Symposium, US Army Communications-Electronics Command, Fort Monmouth, New Jersey.

6. Williams, A. SWAN - An Ada Program for Cost Estimation. AdaIC Newsletter, Sep 1991.

7. DoD Software Reuse Vision and Strategy Document, 15 July, 1992.

3 TRAINING COURSE FOR SYSTEM AND SOFTWARE ENGINEERS

The purpose of this course is to provide system and software engineers with an introduction to system development with domain-specific software reuse. The course introduces the methods necessary to integrate domain-specific software reuse concepts into current system and software development processes by emphasizing domain analysis, generic architecture development, instantiation of the generic architecture, and system composition. An outline of the course description is provided in Appendix A.

3.1 COURSE CHARACTERISTICS

3.1.1 Rationale

This course provides the basis for incorporating domain-specific reuse into system and software development processes. This course introduces the methods necessary to integrate domain-specific reuse concepts into current system and software development processes by providing an overview of domain engineering and detailed guidance on application engineering. The course is intended for use in both Government and industry training and can be tailored for presentation at the university level.

3.1.2 Course Objectives

The course objectives are: to provide a basis for integrating domain-specific reuse into current system and software engineering practices, and to explain how the system/software engineer can use domain analysis products for the modeling, simulation, and prototyping of a system. Although it is necessary to cover domain analysis techniques, the emphasis of this course is in the use of domain analysis products to support domain-specific reuse in application development.

3.1.3 Course Completion Criteria

When the course is presented using all the units identified above, the following products will be produced as proof of course participation:

- A context model for a simple domain,
- A domain model for a simple domain,
- A system architecture from a generic architecture provided by the instructor,
- A system prototype using a domain-specific software reuse library.

3.1.4 Audience Characterization

This course is intended for experienced system and software engineers.

System engineers are concerned with the decomposition of systems, the allocation of software development responsibility for specific system components to software engineers, and with the subsequent composition of software and hardware system components to produce the final system.

System engineers begin with customer-defined goals, requirements and constraints and derive a representation of function, performance, interfaces, design constraints, and information structure that can be allocated to each of the generic system elements.

To accommodate function and performance, defined during system engineering, software engineers must build or acquire a set of software components. The software engineer is responsible for analyzing software requirements, developing or identifying existing software designs and software components, and integrating, testing, and maintaining software components.

3.1.5 Student Readings

1. Arango, G. and Prieto-Diaz, R. Domain Analysis and Software Systems Modeling. IEEE Computer Society Press, May 1991.

2. Biggerstaff, T., Perlis, A. J., Software Reusability, Volume I, Concepts and Models, ACM.

3. Cohen, S., Stanley, J., Peterson, A., Krut, R., Application of Feature Oriented Domain Analysis to Army Movement Control Domain, Software Engineering Institute, (SEI), CMU/SEI-91-TR-28, 30 Sep 1991.

4. Cohen, S., Software Reuse Technology: Feature-Oriented Domain Analysis. SEI Tutorial slides, Mar 1992.

5. Component Provider's & Tool Venders Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-0411/001/00, 15 Mar 1993.

6. DoD Domain Analysis Guidelines, DoD Software Reuse Initiative, Defense Information Systems Agency (DISA) Center for Information Management (CIM), May 1992.

7. Engineers Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04112/001/00, 12 Feb 1993.

8. Frakes, W. B., Gandel, P. B., Representing Reusable Software, Information and Software Technology, Vol. 32, No. 10, Dec 1990.

9. Frakes, W., Prieto-Diaz, R., Arnold, R., Software Reuse, Domain Analysis, and Reengineering, 1992.

10. Hess, J., Novak, W., Carroll, P., Cohen, S., Holibaugh, R., Kang, K., Peterson, A., A Domain Analysis Bibliography, Software Engineering Institute (SEI), CMU/SEI-90-SR-3, June 1990.

11. Holibaugh, R., Joint Integrated Avionics Working Group (JIAWG) Domain Analysis Concepts, 1? Dec 1989.

12. Pressman, R. S. Software Engineering: A Practitioner's Approach. McGraw-Hill, 1992.

13. Prieto-Diaz, R., Domain Analysis: An Introduction, ACM SigSoft, Software Engineering Notes, Vol. 15, No. 2, April 1990

14. Technical Concepts Document. Central Archive for Reusable Defense Software (CARDS), STARS-AC-04107A/001/00, 26 Feb 93.

15. Tracz, W. Software Reuse - Emerging Technology. IEEE Computer Society Press, Sep 1985.

16. Wartik, S., Prieto-Diaz, R., Criteria for Comparing Reuse-Oriented Domain Analysis Approaches, Software Productivity Consortium, 1991.

17. Withey, J., Model-Based Engineering. SEI Tutorial slides, Mar 1992.

18. Yourdon, E. Modern Structured Analysis. Yourdon Press Computing Series, Prentice-Hall, 1989.

19. Yourdon, E. Decline and Fall of the American Programmer, Yourdon Press, 1992.

3.2 OUTLINE OF COURSE CONTENT

3.2.1 Introduction and Rationale

A. DoD Software Reuse Vision and Strategy Overview

1. Considerations:

a. The objective is systematic (planned), not opportunistic reuse,

b. There is no single approach to software reuse,

c. Libraries facilitate but do not enable reuse,

d. Reuse is a process, not an end-product,

e. Domain analysis/models/architectures are the primary focus,

f. Near term cost savings will be offset by infrastructure investments,

g. Ada provides a foundation upon which to base reuse efforts.

2. The Vision:

a. Process-Driven,

b. Domain-Specific,

c. Architecture-Centric,

d. Library-Assisted.

3. The Strategy:

a. Specify domains where reuse opportunities exist,

b. Define reusable products and develop evaluation criteria,

c. Determine ownership criteria of reusable products,

d. Modify the current acquisition process, and integrate reuse into every phase of the system/software life-cycle,

e. Define models which support reuse, to include business decision models, domain models and domain software architectures,

f. Establish metrics collection procedures through which program management can gauge reuse success,

g. Define components standards,

h. Pursue a technology-based investment strategy which identifies, tracks, and transitions appropriate reuse-oriented process and product technologies,

i. Conduct comprehensive training of management and technical personnel,

j. Exploit near-term products and services which facilitate movement to a reuse-based paradigm.

4. Overall Risk is Reduced.

a. Greater uncertainty exists in the costs associated with developing a new component.

b. There is an initial investment associated with reusing an existing component.

B. CARDS Overview

1. Program Goals:

a. Produce, document, and propagate techniques to enable domain-specific reuse throughout the DoD,

b. Develop a Franchise Plan which provides a process for planning domain-specific, library-assisted reuse throughout the DoD,

c. Implement the Franchise Plan with selected users and/or provide a tailored set of services to support reuse,

d. Develop and operate a domain-specific library system and necessary tools.

2. Design with Reuse.

a. Incorporate available reusable assets into system analysis and design.

3. Entry-points of Life-cycle Artifacts.

a. It is important to stress that reusable components include all life-cycle artifacts such as requirements, design, code, and test suites.

C. Evolution of Domain-Specific Software Reuse

1. Domain-specific Software Reuse.

a. A higher return on reuse is realized when requirements are reused rather than just code. The instructor will introduce the student to the concept of domain product reuse. The instructor needs to explain the basic concepts and advantages of domain-specific reuse. This is just a top-level introduction; the detailed presentation follows.

2. Pioneering Software Reuse Projects:

   a. CAMP,

   b. Raytheon Missile Systems Division,

   c. GTEDS Asset Management Program.

D. Payoffs of Domain-Specific Software Reuse:

   Comparison of building a system architecture from scratch as opposed to building it using domain-specific assets

1. Decrease system development time,

2. Increase productivity,

3. Increase quality,

4. Increase maintainability,

5. Increase job performance.

E. Software Reuse Libraries

1. Library as a Reuse Tool,

   The instructor will introduce the reuse library as a tool, not as the reuse solution.

2. Reuse Library Representations,

   The instructor will introduce a brief overview of various methods (e.g., Dewey decimal, semantics, faceted, indexed, and keywords).

3. Search and Retrieval Methods,

   The instructor will have the students consider full text searching, facet searching, keyword searching, knowledge-based searching, and browsing.

4. Available Libraries:

   The instructor will provide the students with an overview of several different types of libraries and the purpose each serves the reuse community.

   a. CARDS,

b. ASSET,

c. DSRS,

d. AdaNET,

e. COSMIC,

f. ASR,

g. Others.

3.2.2 Domain Analysis

The teaching of domain analysis will be complete enough to instill an awareness of domain analysis techniques and their role in systematic software reuse. This section will also build a trust in the products produced by the domain analysis process.

A. Domain Modeling Methodologies:

Most domain modeling methodologies formulate similar models and viewpoints of the domain. These models can often include feature models, functional models, object models, and composition rules. Some of the common modeling techniques used are entity-relationship models, state diagrams, hierarchical models and other structured analysis design techniques. The instructor will review several common structured analysis and modeling techniques. A brief overview of several domain modeling methodologies will be presented.

B. Domain Identification:

It is necessary to choose the right domains to analyze. A domain must be stable, well understood and somewhat static to be a good candidate for domain analysis. A discussion and an illustration of how a domain is chosen will occur.

C. Domain Context Analysis:

The boundaries of the domain must be properly scoped. This process places the domain relative to other domains. This process also defines the external entities and data flows between the external entities and the domain.

Lab: The instructor will provide the students with the name and a brief description of a domain. The students will create a context diagram of the domain specified.

D. Domain Modeling:

The domain model captures the common parts of the domain along with the differences. The data commonality and the control flow for the functionality is captured in a functional model. This functional model is then used to generate the requirements for a generic architecture and the reusable components.

Lab: The students will create a functional model of the domain specified by the instructor.

E. Domain Analysis Products:

Using the models, composition rules, and taxonomy generated by domain analysis, a generic architecture is developed. The generic architecture provides interface specification among the components of the domain model. In developing the generic architecture, software constraints, hardware constraints, performance constraints and general design constraints are all considered. The student must understand the structure of the generic architecture in order to use it.

3.2.3 Integration of Reuse Into System Development Processes

To achieve reuse at the system composition level, the system engineer must understand how the various domain products integrate into the system development process at the requirements definition level. This section will use instructor-provided domain models, system requirements and a domain-specific library to illustrate and support the development of a specific system architecture and a system prototype. This is an intensive hands-on part of the course.

A. Domain-Specific Libraries:

A domain-specific reuse library is used as a tool to model the domain and build a system. The instructor will review some of the earlier library concepts and any additional considerations necessary to build and use a domain-specific library.

B. Instantiation of the Generic Architecture:

Using a generic architecture and reuse library, the student will build a specific system architecture from an instructor-provided generic architecture. The instructor will use this activity to:

1. Demonstrate to the students how to use the domain-specific reuse library.

2. Illustrate to the students how the generic domain analysis products can clarify requirements.

3. Show how the generic domain analysis products can be used to define requirements.

Lab: Design a specific system architecture by using the simple domain model and generic architecture.

C. Generic Artifacts and System Composition:

The software components contained within the software reuse library have been designed in a manner that enables them to be reused without detailed knowledge of the code itself. These components can be assembled to create a prototype. The majority of the course time will be spent doing this exercise.

Lab: The students will perform system composition using the software reuse library, the revised requirements, and the application architecture defined in the previous section.

1. Collect components
2. Record issues, trade-offs and design rationale
3. Link artifacts
4. Integration testing
5. Taking the prototype to the actual system
6. Component adaptation

D. How Reuse Fits their Process:

The students will discuss their current system and software development processes. The instructor will help them identify how reuse can be incorporated into their specific processes.

3.2.4 Summary

The instructor will:

- summarize the important software reuse activities that have been introduced to the student.
- encourage discussion and clarify any issues raised by the students.
- stress the lessons learned during the students' hands-on activities.

This summary will reiterate all the important tasks and how they affect the student.

3.2.5 Lessons Learned

A. Points Confirmed:

1. Improved effectiveness and efficiency of system development,
2. Reduced system development time (testing time and integration),
3. Provided means to rapid prototype systems,
4. Helped clarify requirements (constraints and interfaces),
5. Built trust in domain analysis products,
6. Made job easier.

B. Factors for Failure:

1. Lack of management support for education and training,
2. Lack of company incentives,
3. No quality assurance for evaluating reuse integration into system development,
4. Tools supporting reuse not provided,
5. Lack of system engineer motivation.

3.3 TRAINING INSTRUCTOR

3.3.1 Job Description

The instructor is responsible for conducting a tailored implementation of the recommended course content through a lecture/workshop format.

This is accomplished by completing the following tasks:

- consulting the reference documents listed in Section 3.4,
- updating the recommended outline of course content,
- preparing appropriate training materials,
- conducting the training session,
- developing lab exercises,

- ensuring that the atmosphere of the training session is informal and relaxing, intellectually stimulating, and learner-centered,
- integrating lecture, discussion, and small working groups into the training session,
- leading the course completion criteria workshop, and
- providing an evaluation of the course completion criteria.

3.3.2 Formal Education

The instructor should hold at least a BS degree in system/software engineering (an MS degree is preferred) or a degree in computer science or a closely related field.

3.3.3 Knowledge of Instruction

The instructor should possess an understanding of the principles of learning and the methods of teaching, an ability to apply these principles and methods, and excellent communication skills.

3.3.4 Practical Teaching Experience

The instructor should have at least two years of experience in planning and conducting short courses and should possess teaching experience (at least two years), and should possess experience teaching system and software engineers.

3.3.5 Knowledge of Subject

The instructor should possess:

- experience in the system engineer processes,
- knowledge and an understanding of the impact of integrating reuse into these processes,
- formal training in the form of a workshop or seminar on domain analysis and domain-specific reuse,
- practical experience in reuse library operations, and
- work experience as system engineer or software engineer.

3.4 BIBLIOGRAPHY FOR DEVELOPING THE SYSTEM/SOFTWARE ENGINEERS COURSE

The training instructor should consult the following documents in preparing the tailored implementation of the course content and the training materials:

1. Application Engineering With Domain-Specific Reuse Course Description, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04102B/001/00, 06 March 1993.

2. Architectural Implications For Components Seminar Report, Central Archive for Reusable Defense Software (CARDS), Draft— STARS-VC-B008/000/00. Scheduled for release: 29 January 1994.

3. Basili, V., Caldiera, G., and Cantone, G. A Reference Architecture for the Component Factory. ACM Transactions on Software Engineering and Methodology, Vol. 1 No. 1, Jan 1992.

4. Bell, T. '90s employment: some bad news, but some good. IEEE Spectrum, Dec 1990.

5. Biggerstaff, T. Topics In Reuse and Design. IEEE Video, 1991.

6. Biggerstaff, T., and Perlis, A. Software Reusability Concepts and Models. Vol. I, ACM Press, 1989.

7. Cohen, S. Modeling Software Reuse Technology: Feature Oriented Domain Analysis (FODA). SEI, Carnegie Mellon University, May 1992.

8. Component Provider's & Tool Venders Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-0411/001/00, 15 Mar 1993

9. DiTomaso, N., and Farris, G. Diversity in the High-Tech Workplace. IEEE Spectrum, Jun 1992.

10. Engineers Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04112/001/00, 12 Feb 1993.

11. Fairley, R., and Freeman, P. Issues in Software Engineering Education. Springer-Verlag, 1989.

12. Feiler, P. Configuration Management Models in Commercial Environment. SEI-91-TR-7, SEI, Carnegie-Mellon University, Mar 1991.

13. Frakes, W., Prieto-Diaz, R., and Arnold, R. Software Reuse, Domain Analysis, and Reengineering. Reston, Virginia, 6-8 April 1992.

14. Freeman, P. Tutorial: Software Reusability. IEEE Computer Society Press, 1987.

15. Gause, D. and Weinberg, G. Exploring Requirements Quality Before Design. Dorset House Publishing, 1989.

16. Glass, R. Building Quality Software. Prentice Hall, 1992.

17. Gomaa, H. Kerschberg, L., Bosch, C., Sugumaran, V., Tavakoli, I. A Prototype Software Engineering Environment For Domain Modeling and Reuse, Sixteenth Annual Software Engineering Workshop. NASA Goddard Software Engineering Laboratory, Dec 1991.

18. Hooper, J., and Chester, R. Software Reuse Guidelines. AIRMICS, 13 Dec 1989.

19. Matsumoto, Y. Some Experience in Promoting Reusable Software: Presentation in Higher Abstract Levels. IEEE

20. Transaction on Software Engineering, Vol. SE-10 No. 5, Sep 1984.

21. Pressman, R. Software Engineering A Practitioner's Approach. McGraw-Hill, 1987.

22. Prieto-Diaz, R. Implementing Faceted Classification for Software Reuse. Communications of the ACM, Vol. 34 No. 5, May 1991.

23. Prieto-Diaz, R., and Arango, G. Domain Analysis and Software Systems Modeling. IEEE Computer Society Press, May 1991.

24. Sommerville, I. Software Engineering. Addison-Wesley, 1989.

25. Tomayko, J. Software Engineering Education. SEI Conference 1991, Springer-Verlag, Oct 1991.

26. Tracz, W. Ada Reusability Efforts: A Survey of the State of the Practice. Proceedings of the Joint Ada Conference, Fifth National Conference on Ada Technology and Washington Ada Symposium, US Army Communications-Electronics Command, Fort Monmouth, New Jersey.

27. Tracz, W. Tutorial: Software Reuse: Emerging Technology. IEEE Computer Society Press, 1990.

28. Tracz, W., and Coglionese, L. Domain-Specific Software Architecture Engineering Process Guidelines. ADAGE-IBM-92-02, Ver. 0.1, IBM Corporation, 17 Mar 1992.

29. Withey, J. Model-Based Engineering. SEI Tutorial slides, Mar 1992.

30. Yourdon, E. Modern Structured Analysis. Yourdon Press Computing Series, 1989.

4 TRAINING COURSE FOR UNIVERSITIES

The main objective of this course is to provide university professors with an understanding of software reuse, and to identify potential avenues for integrating domain-specific reuse into university courses and curricula.

4.1 COURSE CHARACTERISTICS

4.1.1 Rationale

Reuse is a recurring topic which must be addressed throughout the computer science curriculum. The incorporation of reuse into the current curriculum will provide students entering the work force with an awareness of the benefits and application of reuse. This course will provide support for the integration of reuse concepts into university curricula.

4.1.2 Course Objectives

The learning objectives for this course are to provide professors with an understanding of reuse, and domain-specific reuse; and to identify potential avenues for integrating domain-specific reuse into their courses and curricula.

4.1.3 Course Completion Criteria

The course completion criteria allows for small interactive working groups to apply concepts and techniques identified by the course's content areas, producing products as proof of completing the training course. Upon completion of the training course the participants will be able to:

- understand where analysis techniques, reuse concepts, and model engineering fit into the current computer science curriculum,
- identify courses where reuse may be integrated,
- outline specific procedures for implementing reuse across the curriculum, and
- formulate a "strawman" curriculum with course descriptions.

4.1.4 Audience Characterization

This course is intended for college and university faculty who possess an interest in:

- the concept of reuse,

- remaining competitive with other universities,
- new research topics, and
- integrating reuse into their software engineering instruction.

The intended audience is responsible for curriculum development, appropriations and developing hiring guidelines. It is assumed that the participants have a software engineering background.

4.1.5 Student Readings

1. Arnold, R. S. Heuristics for Salvaging Reusable Parts from Ada Source Code. Ada Reuse Heuristics, 90011-N, Software Productivity Consortium, Mar 1990.
2. Braun, C. Ada Reusability Guidelines. Softech, Apr 1985.
3. Computing Curricula 1991 - Report of the ACM/IEEE-CS Joint Curriculum Task Force. ISBN 0-8186-2220-2, Mar 1991.
4. Frakes, W. B. An Empirical Framework for Software Reuse Research. Proceedings of the Third Workshop on Methods and Tools for Reuse, No 9014, Syracuse University CASE Center, 1990.
5. Freeman, P. Reusable Software Engineering: Concepts and Research Directions. Workshop on Reusability in Programming, ITT Programming, Sep 1983.
6. McClure, C. The Three R's of Software Automation: Reengineering, Repository, Reusability. Prentice Hall, 1992.
7. Pressman, R. S. Software Engineering: A Practitioner's Approach. McGraw Hill, 1992.

4.2 OUTLINE OF COURSE CONTENT

The following is an outline of the recommended course content for universities.

4.2.1 Reuse as Part of the Solution

A. The Dilemma:

1. Software industry:

a. Increasing software costs versus hardware costs,

b. Increasing complexity of software,

c. Increasing development time,

d. Shortage of experienced personnel.

2. Customer:

a. Demand for high quality products,

b. Demand for low risk products,

c. Demand for automated programming support,

d. Demand for customized software packages.

B. The Solution:

1. What is Reuse?

a. The process of incorporating into the life-cycle of a software system any preexisting components (e.g, requirements, design, code, executables).

b. Examples.

2. Why Reuse?

a. Reduces schedule,

b. Increases productivity:

Amplifies programming capabilities,

Reduces the amount of documentation and testing.

c. Increases quality:

Software is well designed (for reuse),

Software is well documented (standard),

Software is well tested (certified),

Software is well understood (functionality), Concepts are manifested in rapid prototyping.

d. Increases reliability,

e. Reduces maintenance costs,

f. Overall, reuse allows for more system development within the same time frame,

g. Reduces development costs.

3. Evolution of Reuse:

a. Early reuse projects,

b. Reuse successes,

c. Current approaches and concepts:

Components and wide spectrum reuse,

Composition versus generation,

Design for reuse,

Design with reuse,

Life-cycle importance of reuse.

4.2.2 Introduction to Domain-Specific Reuse

A. What is a Domain?

1. A domain is a set of common capabilities and data which constitute a set of current and future systems in a particular application area, such as:

a. A business area,

b. A software business area,

c. A software intensive application area,

d. An application area for which similar software systems have been built.

2. Examples.

B. What is Domain-Specific Reuse?

1. Reuse of ideas, knowledge, artifacts, personnel, and components in an existing domain.

2. Examples.

C. Why Domain-Specific Reuse?

1. Reuse is more effective in a narrow, well defined domain where similar systems are built.

2. Examples.

D. Domain Selection Checklist:

1. Define strategy.

2. Select domain analysis resources.

3. Identify boundaries/scope of domain.

4. Select mature, stable, well-defined domain.

5. Define predictable technology.

6. Perform market evaluation.

7. Identify available domain expertise.

8. Conduct readiness cost and benefit analysis.

E. Existing Technological Support:

1. STARS efforts,

2. CARDS,

3. PRISM,

4. DSRS,

5. Others.

4.2.3 Reuse and the Software Life-Cycle

A. Design for Reuse.

Build into the design and implementation the flexibility, variations and adaptability to create a reusable component.

B. Design with Reuse.

Incorporate available reusable assets into system analysis and design.

C. Entry-points of Life-cycle Artifacts.

It is important to stress that reusable components include all life-cycle artifacts such as requirements, design, code, test suites.

D. Reengineering:

1. When to reengineer,
2. Economic considerations,
3. Data reengineering and software reengineering,
4. Benefits of reengineering:
    a. Reduced maintenance,
    b. Upgrading software along with upgrading software engineering practice,
    c. Easier documentation and testing.
5. Incremental reengineering.

E. CASE:

1. Features of CASE tools,
2. CASE Issues,
3. Sample CASE tools,
4. CASE tools as a research topic.

4.2.4 Considerations When Integrating Reuse

A. Economic Issues.

A university must consider the costs associated with incorporating reuse.

1. How much will it cost to implement reuse into the curriculum?
2. Component availability.
3. Library maintenance and availability.
4. Is new hardware required?
5. Network connection cost.
6. What are the alternatives?

B. Feasibility Analysis.

A university must decide if it is feasible to incorporate reuse into their current curriculum.

1. Is the reuse program supported by administration?
2. Does the school have the resources to implement the program?
3. Does the department have personnel with the appropriate training?
4. What is the scope of reuse in the curriculum?

C. Administrative/Departmental Issues.

1. Departmental support:
   - a. Professors,
   - b. Instructors,
   - c. Assistant chairman,
   - d. Chairman.
   - e. Does the department have a long term commitment to reuse?

2. Administrative Support:
   a. Dean of college,
   b. Dean of academic affairs.
   c. Is the administration aware of the benefits of reuse?
3. Technical Aspects:
   a. Can existing hardware be used?
   b. What networks need to be accessed?
   c. What library mechanisms are required?
   d. What other software is required (CASE tools)?
   e. Alternatives.
4. Incremental In-House Reuse:
   a. Software engineering,
   b. Design for reuse and data abstraction,
   c. System analysis,
   d. Domain analysis,
   e. Put components designed for reuse into in-house reuse library,
   f. Design with reuse using existing in-house reuse library,
   g. Introductory courses and reuse concepts.

4.2.5 Continuing Education/Research

A. Does the staff have interest in reuse as a research topic?

B. Is there funding available from Government/industry to support research?

4.2.6 Summary of Content

A. Gain firm commitment from staff and administration,

B. Assess benefits of introducing reuse into the curriculum,

C. Identify and resolve impediments to the introduction of reuse into the curriculum.

4.3 TRAINING INSTRUCTOR

4.3.1 Job Description

The training instructor is responsible for conducting a tailored implementation of the recommended course content through a lecture/workshop format. This is accomplished by completing the following tasks:

- consulting the documents listed in Section 4.4;
- updating the recommended outline of course content;
- preparing appropriate training materials;
- estimating the duration of the training session based on a blend of optimum cost-effectiveness and covering the course content thoroughly;
- conducting the training session;
- seeing that the atmosphere of the training session is informal and relaxing, intellectually stimulating, and learner-centered;
- integrating lecture, discussion, and small working groups into the training session;
- leading the course completion criteria workshop; and
- providing an evaluation of the course completion criteria.

4.3.2 Formal Education

The instructor should hold a PhD in computer science, software engineering, system engineering, or a closely related field.

4.3.3 Knowledge of Instruction

The instructor should possess an understanding of the principles of learning; the methods of teaching; an ability to apply these principles and methods; and excellent communication skills.

4.3.4 Practical Teaching Experience

The instructor should possess teaching experience (at least two years), experience in planning short courses, and in addressing university faculty.

4.3.5 Knowledge of Subject

The instructor should possess formal training in the form of a workshop or seminar on domain analysis and domain-specific reuse, and work experience in software development, domain analysis, and reengineering.

4.4 BIBLIOGRAPHY FOR DEVELOPING UNIVERSITY COURSE CONTENT

The training instructor should consult the following documents in preparing the tailored implementation of the course content and the training materials.

1. Matsumoto, Y. Some Experience in Promoting Reusable Software: Presentation in Higher Abstract Levels. IEEE

2. Transaction on Software Engineering, Vol. SE-10 No. 5, Sep 1984.

3. Pressman, R. Software Engineering A Practitioner's Approach. McGraw-Hill, 1987.

4. Tomayko, J. Software Engineering Education. SEI Conference 1991, Springer-Verlag, Oct 1991.

5. Tracz, W. Ada Reusability Efforts: A Survey of the State of the Practice. Proceedings of the Joint Ada Conference, Fifth National Conference on Ada Technology and Washington Ada Symposium, US Army Communications-Electronics Command, Fort Monmouth, New Jersey.

5 EXECUTING THE TRAINING PLAN

There are two primary methods by which the Training Plan can be effectively executed.

5.1 ONE COURSE AT A TIME

The first method is to conduct each course independent of the other courses, i.e., one course at a time. This is accomplished by appointing a Training Supervisor to oversee the training course's management and a Training Instructor to conduct the actual training session.

All of the necessary information for pursuing this strategy is encapsulated for each training course in its respective section in this document. This allows for each training course to be independently removed from the Training Plan and forwarded to perspective implementors without any required knowledge of the other training courses.

5.2 INTEGRATED PACKAGE

The second method is to conduct each course as an integral part of a total package. This requires appointing a Training Executor to oversee the top-level management of this strategy.

The Training Executor identifies organizations that have a working relationship to enable more than one organization to attend the same class. The Training Supervisors and Training Instructors function in the same manner as they would in conducting independent versions of their courses. The Executor then formulates a coherent set of instructional courses for each audience.

Of the two methods for executing the Training Plan, the integrated package will have a stronger impact on integrating reuse and a higher potential of accomplishing a shift within DoD and industry to reuse-based domain-specific software development.

The necessary information for pursuing this strategy is a combination of each of the three training courses and the administrative functionality of the Training Executor. Each training course becomes a tailored implementation of the Training Plan based not only on the target audience but also on considerations of the other audiences.

5.3 TRAINING EXECUTOR

The Training Executor's functionality is important for the successful implementation of the Training Plan as an integrated package. To ensure proper and efficient implementation of the Training Plan, the Training Executor is responsible for completing the following tasks:

- Identify DoD organizations, DoD contractors, and universities that have previously established a working relationship with each other. These audiences could also be identified from a geographical perspective to assist in building such working relationships;

- Select an appropriate training facility available to all audiences, if applicable;
- Coordinate the efforts of Training Supervisors;
- Assure the correct implementation of the Training Plan based on the established working relationship between DoD organizations, DoD contractors, and universities; and
- Monitor the impact of the training across the three audiences.

5.4 TRAINING SUPERVISOR

The training supervisor's functionality is important for the successful execution of the training course. The time period for completing the following tasks is approximately 6-8 weeks. The Training Supervisor is responsible for:

- describing the intent of the course to the Training Instructor;
- determining the optimum number of participants;
- scheduling the training sessions by coordinating the facility's availability with the Training Instructor;
- addressing food, lodging, and transportation;
- registering the participants;
- forwarding materials to each participant;
- coordinating all efforts between the Training Instructor, facility staff, and Training Executor;
- assuring the correct implementation of the recommended course content by the Training Instructor;
- securing training equipment;
- reproducing training materials;
- administering evaluations of the training;
- producing a lessons learned document containing information that may be used in updating the Training Plan and/or the course content at a later date; and

- documenting the impact of the training by tracking requests for further training and requests for further information.

5.5 TRAINING FACILITY

The style of the classroom must be established. Some factors to consider:

- lecture versus conference style room,
- seating capacity,
- computer resources, and
- proximity to convenient facilities.

Training devices appropriate to the types of training materials must be available.

5.6 PRICE WORKSHEET

The following is a Price Worksheet to assist in predicting dollar amounts for conducting the training courses.

A. Price Worksheet

1. Administration
2. Instructional time/travel costs
    a. Supervisor
    b. Instructor
    c. Technician
    d. Participant time/travel costs
3. Scheduling training session
4. Registration of participants
5. Site set-up
6. Refreshments

7. Communications

B. Course Materials

1. Preparation for training instructor

    a. Agenda

    b. Course outline

    c. Handouts

    d. Transparencies/overheads

2. Reproduction for participants

    a. Paper copies of slides

    b. Lab worksheets

    c. Handouts

    d. Course evaluation forms

    e. Proceedings

C. Support Costs

1. Facilities

2. Equipment

APPENDIX A - Application Engineering with Domain Specific Reuse

A.1 INTRODUCTION

A.1.1 PURPOSE

This course provides the basis for incorporating domain-specific reuse into system and software development processes. This course introduces the methods necessary to integrate domain-specific software reuse concepts into current system and software development processes by providing an overview to domain engineering and detailed guidance on application engineering. The course is intended for use in both Government and industry training and can be tailored for presentation at the university level.

A.1.2 CARDS PROGRAM MISSION

The Central Archive for Reusable Defense Software (CARDS) Program is a concerted Department of Defense (DoD) effort to transition advances in the techniques and technology of library-centered, domain-specific software reuse into mainstream DoD software procurements. There are three key elements to the CARDS approach:

1. Apply domain-specific reuse techniques and technology to produce an operational library for command centers

2. Develop and transition, through a Franchise Plan, the "knowledge" for domain-specific reuse to the DoD and DoD Software Development Industry

3. Develop and transition a Training Plan and training courses as a vehicle for enhancing the acceptance of the Franchise Plan.

4. The domain-specific reuse knowledge gained during the CARDS effort will be conveyed via a Franchise Plan and three sets of documents: Reuse Adoption Handbooks, CARDS library operation and maintenance related documents, and training and educational material.

A.1.3 RELATIONSHIP TO OTHER CARDS DOCUMENTS

The Franchise Plan provides a description of reuse processes and instructions for tailoring development processes to effect domain-specific reuse. It describes, in precise steps, a scenario for an organization to establish a domain-specific reuse capability. The Reuse Adoption Handbooks consist of the Component and Tool Developer's, Acquisition, Direction Level, and Engineer's Handbooks. Together these four handbooks address software development, program management, and executive planning. The Component and Tool Developer's Handbook addresses the development of reusable software components. Software industry vendors supporting Government acquisitions are provided with guidance for developing/creating domain-specific reusable components and tools supporting reuse. The goal of this Handbook is to stimulate the development and commercialization of large scale components and tools for vertical domains. [CARDS93b]. The Acquisition Handbook assists Government Program Managers and their support staff in incorporating software reuse into the acquisition and maintenance portions of the life-cycle process. The Acquisition Handbook provides guidance in planning the acquisition strategy, contract award, managing the effort, and follow-on support [CARDS93c]. The Direction Level Handbook offers a framework to assist Government acquisition executives in establishing plans to manage software reuse across their systems. Importance is placed on the policy and business issues (e.g., regulations, incentives, funding, cost/benefit, education and training, and ownership of components) that act as the support structure for reuse [CARDS92a]. The Engineer's Handbook provides guidance to Government System Program Office (SPO) Engineers on envisioned changes to their duties and responsibilities as domain-specific software reuse becomes incorporated into mainstream DoD system/software acquisition and engineering processes. [CARDS93a].

Although some of the CARDS library operation and maintenance documents are specific to the CARDS library, they can be used by other organizations to learn how reuse was implemented in the command and control domain. These CARDS documents address the library's operations procedures, the technical concepts, project management plans, as well as describing the domain model. The CARDS training effort includes a Training Plan, course outlines, and sample course materials relating to topics included in each Reuse Adoption Handbook. They are geared to educate the software professional and support the reduction of cultural barriers to reuse. They can be tailored to meet the needs of varying audiences. This course has been developed as a means of conveying the concepts and techniques introduced in the Engineer's Handbook.

A.1.4 DOCUMENT ORGANIZATION

The course description begins by providing the reader with a description of the intended course audience. Section 2 characterizes the student and identifies required prerequisite knowledge. Section 3 provides the reader with a guideline for selecting an instructor to present the course. Section 4 provides a course overview. Course content for each of the units is described in detail in Section 5. Included with this course description is a sample set of course materials. The materials can be used as they are, or modified to suit the needs of the presenting organization.

A.2 STUDENT

A.2.1 STUDENT CHARACTERIZATION

This course is intended for experienced system and software engineers.

A.2.1.1 System Engineers

System engineers are concerned with the decomposition of systems, the allocation of software development responsibility for specific system components to software engineers, and with the subsequent composition of software and hardware system components to produce the final system [TOMA91]. System engineers begin with customer-defined goals, requirements and constraints and derive a representation of function, performance, interfaces, design constraints, and information structure that can be allocated to each of the generic system elements [PRES87].

A.2.1.2 Software Engineers

To accommodate function and performance, defined during system engineering, software engineers must build or acquire a set of software components [PRES87]. The software engineer is responsible for analyzing software requirements, developing or identifying existing software designs and software components, and integrating, testing, and maintaining software components.

A.2.1.3 Student Prerequisite Knowledge

Successful completion of this course requires that the student possess prior experience participating in a complex software development project (e.g., at least 100K Source Lines of Code). Students must possess knowledge of systems analysis and requirements analysis, and knowledge of structured analysis design techniques and/or object-oriented design techniques to the depth presented in a one semester undergraduate advanced software engineering course. The student should be literate in the programming language(s) used for the course demonstration material.

A INSTRUCTOR

A.3.1 JOB DESCRIPTION

The instructor is responsible for conducting a tailored implementation of the recommended course content through a lecture/activity format. This is accomplished by completing the following tasks:

1. Consulting the reference documents,
2. Reviewing the recommended outline of course content and choosing appropriate levels of content based on audience and time,
3. Preparing appropriate training materials,
4. Setting up activities,
5. Ensuring the atmosphere of the training session is informal, relaxing, intellectually stimulating, and student-centered,

6. Integrating lecture, discussion, and small working groups into the training session,

7. Conducting the training session,

8. Leading the course completion criteria activities,

9. Providing an evaluation of the course completion criteria.

A.3.2 FORMAL EDUCATION

The instructor should hold at least a BS degree in software/system engineering (an MS degree is preferred) or a degree in computer science or a closely related field.

A.3.3 KNOWLEDGE OF INSTRUCTION

The instructor should possess an understanding of the principles of learning and teaching methodologies, an ability to apply these principles and methods, and excellent communication skills.

A.3.4 PRACTICAL TEACHING EXPERIENCE

The instructor should have at least two years experience teaching short courses, should have a minimum of two years experience teaching at the collegiate level, and should possess experience addressing system and software engineering personnel.

A.3.5 KNOWLEDGE OF SUBJECT

The instructor with an MS degree should possess a minimum of five years practical work experience in the system and software engineering processes, or ten years experience with a BS degree. The instructor must be knowledgeable in software reuse and understand the impact of integrating software reuse into system and software engineering processes. The instructor must have completed formal training in the form of a workshop or seminar on domain analysis and domain-specific software reuse. For presentation of the course to DoD personnel, the instructor should also have knowledge of general DoD engineering and procurement standards as well as the various DoD mandates effecting system development and maintenance.

A.3.6 SKILL IN PERFORMANCE

The instructor should be able to demonstrate the ability to perform the tasks required in the practical application of the training course and demonstrate the ability to conduct domain analysis activities and perform application engineering using the products of domain engineering.

A.4 COURSE OVERVIEW

A.4.1 COURSE BREAKDOWN

The content for this course has been broken into seven units. These units can be presented independently or contiguously. Students should attend units in the order described; however, units can be skipped if students possess the knowledge to be presented. The units comprising this course are as follows:

Unit 1: Introduction and Rationale. Anticipated Duration: 2 hours

An introduction to the DoD Software Reuse Vision and Strategy [DOD92] is provided. An overview of the CARDS Program is presented and DoD reuse coordination efforts are discussed.

If the presenting organization is not CARDS, an appropriate introduction should replace this unit. It is recommended that the introduction include an overview of the presenting organization and its rationale for teaching domain-specific reuse.

Unit 2: Software Reuse Concepts. Anticipated Duration: 2 hours

An overview of software reuse is presented. This unit addresses why reuse should be considered and how long reuse has been practiced. Reusable components are defined. Differences in reuse approaches are discussed (e.g., ad hoc, opportunistic, systematic, large scale reuse, small scale reuse). Finally, potential risks and benefits associated with reuse are presented.

Unit 3: Domain-Specific Reuse. Anticipated Duration: 2 hours

A brief history of domain-specific reuse begins this unit. An overview of domain engineering is provided and the products of domain engineering are discussed. Application engineering within the domain-specific reuse framework is introduced and explored. Students learn how domain engineering products are incorporated into every software development activity from requirements analysis through maintenance.

Unit 4: Domain Analysis Overview. Anticipated Duration: 4 hours

This unit provides an overview of domain analysis activities and products. Students receive hands-on experience performing context analysis and developing a domain model. This is a high-level domain analysis unit. It is not intended for students who will actually perform domain analysis tasks. Instead it has been developed to provide students who will use domain analysis products with a better understanding of what the products are and how they were developed.

Unit 5: Software Reuse Libraries. Anticipated Duration: 2 hours

An overview of component-based and model-based libraries is provided. Students are introduced to various types of library representations and mechanisms, and certification and qualification of library components. The intent of this unit is to build trust in the students for reusable components. Students will learn to use an actual reuse library.

Unit 6: Application Engineering. Anticipated Duration: 8 hours

This unit is intended to teach students how to perform application engineering using the products of domain engineering. Students perform requirements analysis and application prototyping using a reuse library. The impact of domain-specific reuse on system maintenance activities is discussed. The format of this unit is both lecture and activity.

Unit 7: Reuse Barriers. Anticipated Duration: 1 hour

Barriers to reuse are addressed in this unit. For each barrier discussed, the instructor presents possible solutions students can use to overcome these obstacles. Students will explore the potential barriers they can expect to encounter in their work environment and how they can plan to overcome them.

A.4.2 COURSE COMPLETION CRITERIA

When the course is presented using all the units identified above, the following products will be produced as proof of course participation:

1. A context model for a simple domain,
2. A domain model for a simple domain,
3. A system architecture from a generic architecture provided by the instructor,
4. A system prototype using a domain-specific software reuse library.

A.4.3 COURSE TAILORING

The following are examples of how this course can be tailored to particular audiences.

A.4.3.1 DoD Personnel With No Reuse Experience

When the course is presented to DoD personnel with no reuse experience, all seven units, defined above, will be presented. All activity examples will be DoD based, and the modeling performed in Unit 4 will use IDEF0 and IDEF1X modeling languages.

A.4.3.2 University Graduate Students

When the course is presented to university graduate students, all seven units will be presented. The course duration will change to fifteen weeks, as follows:

Unit One: Introduction and Rationale will be presented in 1 day.

Unit Two: Software Reuse Concepts will be presented in 1 week.

Unit Three: Domain-Specific Reuse will be presented in 1 week.

Unit Four: Domain Analysis Overview will be presented in 2 weeks.

Unit Five: Software Reuse Libraries will be presented in 1 week.

Unit Six: Application Engineering will be presented in 8 1/2 weeks.

Unit Seven: Reuse Barriers will be presented in 1 week.

The activity defined in Unit 6 will be extended to a real-world, large-scale development effort. Students will perform all aspects of application engineering from requirements analysis through maintenance using existing domain engineering products. Students will identify the differences between software development with and without domain-specific reuse. It is essential that students have a working knowledge of structured analysis design techniques and/or object oriented design techniques.

A.5 ANNOTATED COURSE OUTLINE

A.5.1 UNIT 1 - INTRODUCTION AND RATIONALE

A.5.1.1 Learning Objectives

This introduction has been developed for presentation of the course by CARDS. Since CARDS is a concerted DoD effort, it is necessary for students to understand the software reuse goals of the DoD. The objectives of this lecture are to provide students with an understanding of the DoD Software Reuse Vision and Strategy document, the CARDS Program goals and products, and the role of CARDS in the DoD's current reuse efforts. If this course is being presented by an organization other than CARDS, this unit can be replaced in its entirety, or the CARDS overview may be removed and replaced with an overview of the presenting organization. Upon completion of this lecture the student will be able to:

1. Explain DoD's Software Reuse Vision and Strategy,
2. Define process-driven,
3. Define domain-specific,
4. Define architecture-centric,
5. Define library-assisted,
6. Define the goals of the CARDS Program,
7. List the major activities, products, and services of the CARDS Program,
8. Describe the role CARDS plays in DoD's reuse efforts,

9. Explain what is meant by library interoperability,

10. Explain the interoperability coordination among the major DoD reuse efforts.

A.5.1.2 Lecture Content Area

A. DoD Software Reuse Vision and Strategy Overview

The DoD Software Reuse Vision and Strategy [DOD92] describes an initiative which will make a reuse-based paradigm the preferred alternative for developing and supporting software. Students will learn what this document is and how it may effect the way they currently practice software development.

1. Considerations

There are infrastructure investments associated with establishing an effective reuse program. The DoD Software Reuse Vision and Strategy considers the following [DOD92]:

a. The objective is systematic (planned), not opportunistic reuse,

b. There is no singular approach to software reuse,

c. Libraries facilitate but do not enable reuse,

d. Reuse is a process, not an end-product,

e. Domain analysis/models/architectures are the primary focus,

f. Near term cost savings will be offset by infrastructure investments,

g. Ada provides a foundation upon which to base reuse efforts.

2. The Vision

"To drive the DoD software community from its current 're-invent the software' cycle to a process-driven, domain-specific, architecture-centric, library-assisted way of constructing software"[DOD92].

a. Process-Driven

Software development is done in accordance with well defined, repeatable processes that are enforced through management policies and for which, at a minimum, definition and guidance are provided in the software engineering environment [STARS92a].

b. Domain-Specific

A domain is an area of activity or knowledge containing applications which share a set of common capabilities and data. Domains should be exploited to support the reuse of a wide spectrum of products, including but not limited to requirements, architectures, designs, algorithms, subsystems, test plans, and development processes.

c. Architecture-Centric

Generic architectures will be used for well established domains to develop generic software components. The goal is to achieve "black-box" reuse. Black-box reuse is realized when components can be interchanged simply by removing one and replacing it with another.

d. Library-Assisted

A network of interconnected, "interoperable" reuse libraries will be developed which will be used to store, manage and retrieve reusable components within and across domains.

3. The Strategy

"The strategy to realize this vision is based on systematic reuse: where opportunities are predefined and a process for capitalizing on those opportunities is specified" [DOD92]. The strategy is broken down into the ter following elements:

a. Specify domains where reuse opportunities exist,

b. Define reusable products and develop evaluation criteria,

c. Determine ownership criteria of reusable products,

d. Modify the current acquisition process, and integrate reuse into every phase of the system/software life-cycle,

e. Define models which support reuse, to include business decision models, domain models and domain software architectures,

f. Establish metrics collection procedures through which program management can gauge reuse success,

g. Define component standards,

h. Pursue a technology-based investment strategy which identifies, tracks, and transitions appropriate reuse-oriented process and product technologies,

i. Conduct comprehensive training of management and technical personnel,

j. Exploit near-term products and services which facilitate movement to a reuse-based paradigm.

B. CARDS Overview [WAL92]

The Central Archive for Reusable Defense Software (CARDS) Program is a concerted Department of Defense (DoD) effort to transition advances in the techniques and technology of architecture-based, library-assisted, domain-specific software reuse into mainstream DoD software procurements. Students will learn what the goals of the CARDS Program are, what activities CARDS is involved in, and what products and services CARDS has to offer the DoD software community. This is intended to be a brief overview. Students will be directed to specific documents for additional information.

1. Program Goals

a. Produce, document, and propagate techniques to enable domain-specific reuse throughout the DoD,

b. Develop a Franchise Plan which provides a blueprint for institutionalizing domain-specific, library-assisted reuse throughout the DoD,

c. Implement the Franchise Plan with selected organizations and/or provide a tailored set of services to support reuse,

d. Develop and operate a domain-specific library system and necessary tools.

2. CARDS Products

a. Franchise Plan

A knowledge blueprint is expressed as a Franchise Plan which references a series of handbooks, library documentation, and training materials, each targeted to a particular audience which needs to be involved in making reuse happen in DoD software procurements. The Franchise Plan provides the necessary structure for implementing a reuse infrastructure in a DoD organization.

b. Reuse Handbooks

The Acquisition Handbook [CARDS93c] is directed toward program n..nagers and contracting and legal professionals. It encourages software reuse during the contractual and acquisition portion of the software development life-cycle. The Direction Level Handbook [CARDS92a] is directed toward top-level managers. It acts as a reference to assist managers in implementing software reuse across programs within a given mission area. The Engineer's Handbook [CARDS93a] is directed towards system and software engineers and other technical personnel. It provides contractor and DoD personnel with software reuse development methods and techniques to be integrated into their current software engineering processes. The Component and Tool Developer's Handbook [CARDS93b] provides a technical basis for the creation of components and tools for domain-specific reuse libraries.

c. Technical Documents

Documents have been developed detailing CARDS experiences in constructing and operating a domain-specific reuse library. These documents include the Technical Concepts Document (TCD)[CARDS94a], the Library Operations Policies and Procedures (LOPP)[CARDS92b], and the Domain Model Document (DMD)[CARDS92c].

d. Training

Education is the foundation for integrating new concepts into an existing process. During Phase II of the CARDS Program, this Training Plan [CARDS94b] was developed for DoD and DoD industry personnel, and undergraduate and graduate computer science and software engineering students. In addition, a training course and materials for system and software engineers was developed.

e. Command Center Library

Using the Generic Command Center Architecture produced by the Portable, Reusable, Integrated Software Modules (PRISM) Program, a library model based on the Software Technology for Adaptable, Reliable Systems (STARS) Reusability Library Framework (RLF) technology has been created and the PRISM components are being added to the library. The Command Center Library demonstrates the potential of library-assisted, domain-specific reusability based upon domain-specific softv.are architectures. The techniques used to develop this Library can be applied to new domains. Additional domains are planned. An informational hotline is available 8 am to 5 pm EST at 1-800-828-8161.

3. CARDS Services

The CARDS Program will provide interested parties with technical expertise in the field of domain-specific, library-assisted software reuse. In the future, CARDS will provide domain engineers with an implementation framework for reuse libraries in their domain of interest. CARDS will also guide application engineers through the selection of alternative components based on generic architectures. Planned CARDS consulting services include reuse adoption support, domain analysis/modeling, complete site and operation survey, feasibility studies, and operational hardware and software installation.

C. Interoperability Coordination Among DoD Reuse Efforts

CARDS is not a program in isolation, but one of several related reuse initiatives. Students will be provided with an overview of several of the major DoD domain-specific operational reuse library efforts and how they are coordinating to achieve interoperability. Students will learn about the Reuse Library Interoperability Group (RIG) and why CARDS is a participating member. Students will gain an understanding of how CARDS, the STARS Asset Source for Software Engineering Technology (ASSET), and the Defense Information Systems Agency (DISA) Center for Information Management (CIM) Defense Software Repository System (DSRS) are coordinating.

1. Interoperability

To meet the DoD Software Reuse Vision, the initial steps towards interoperability have been taken by CARDS and ASSET. Plans for the future include full interoperability between the CARDS Command Center Library, ASSET, and DSRS.

2. Reuse Library Interoperability Group (RIG)

The RIG is a volunteer, consensus-based organization composed of members from Government, academia, and private industry. Its job is to draft standards for the interoperability of reuse libraries in the areas such as nomenclature, interchange protocols, and software component exchange formats. Most DoD and other Government reuse library programs are members, such as STARS, ASSET, CARDS, and DISA/CIM [PAYTON92].

3. Asset Source for Software Engineering Technology (ASSET)

ASSET's mission is to establish a national resource for the advancement of software reuse across DoD and to foster a software reuse industry in the United States by providing a distributed support system. ASSET has established a reuse library and a National Software Technology Transfer center [ASSET91].

4. Defense Information Systems Agency/Center for Information Management (DISA/CIM)

The mission of the DISA/CIM Program is to increase productivity and quality through software reuse. Toward this end the Program provides a reuse repository, the Defense Software Repository System (DSRS). Some of the goals of the Program are to implement a software reuse environment which provides support for the entire software development life-cycle, cultivate reuse skills, and produce solutions for significant reuse management issues [DISA92a].

A.5.2 UNIT 2 - SOFTWARE REUSE CONCEPTS

A.5.2.1 Learning Objectives

The objectives of this lecture are to provide the students with an understanding of software reuse and an appreciation of the potential benefits of reuse. Upon completion of this lecture the student will be able to:

1. Explain why software reuse is receiving increased attention,
2. Discuss why software reuse is not a new concept,
3. Define reusable components,
4. Define ad hoc reuse,
5. Define opportunistic reuse,
6. Define systematic reuse,
7. Discuss the benefits of systematic reuse,
8. Provide an example of the benefits of large scale reuse over small scale reuse,
9. Explain why exploiting reuse early in the software development life-cycle can result in greater benefits,
10. Explain potential risks associated with software reuse,
11. List three incentives for integrating software reuse into software development processes,
12. Explain how reuse can reduce the level of effort required during maintenance activities.

A.5.2.2 Lecture Content Area

Software reuse is the reapplication of domain knowledge, development experience, design decisions, architectural structures, requirements, designs, code, and documentation from existing systems to an emerging system in an effort to reduce the costs associated with software development and maintenance [BIGG89]. Cost includes manpower (skills and availability), risks to the schedule, and software and hardware budgets.

A. Why Reuse?

There are many reasons why software reuse is moving from research into practice. The instructor will provide the students with an understanding of some of the reasons why software reuse should be given serious attention.

1. The Demand For High Quality Software Is Increasing

   Software of poor quality is very difficult and expensive to maintain [HOOP89]. Components that have been previously tested and used in a project will have fewer errors. As a component is reused and maintained, quality and reliability can be expected to improve [SPC92].

2. The Demand For Improved Productivity Is Increasing

   The DoD plans to greatly increase productivity by the year 2000 [ANTHES92]. Reusing proven software components can help meet this demand. Taking advantage of efforts previously spent on developing software will decrease the resources required for development of new applications and will result in improved productivity and reduced schedules [SPC92], [NATO].

3. The Demand For Software Is Increasing Faster Than The Supply Of Software Developers

   United States universities are producing far fewer people with advanced computer science degrees than are required by industry [YOUR92]. Reuse can help overcome this deficiency. Experienced engineers can create reusable components which encapsulate their knowledge. Less experienced engineers (of which there are generally more) can use these components to create more new systems [SPC92].

4. Advances In Hardware Are Driving Advances In Software

   Hardware technology continues to improve at a steady rate of 20 to 30 percent annually, compounded. Hardware advances drive software application development to take advantage of new hardware capabilities. Reuse of existing components can reduce the time required to meet the demand for new applications [YOUR92].

5. Demand For Early Proof-of-Concept Prototypes Is Increasing

   Requests for Proposal (RFPs) requiring reuse and live test demonstrations are becoming more common. Reuse of existing components can provide the means for creating early proof-of-concept prototypes which can be used to demonstrate an organization's capability to meet customer requirements [SPC92].

6. Software Reuse Is Not New

   Software developers have been practicing software reuse since as far back as the 1950's, when Wilkes, Wheeler and Gill first recognized the importance of reusable subprogram libraries. From the 1950's through the 1980's reuse has been researched and ideas have been formalized. What is new in the 1990's is the methodology being developed to allow developers to leverage the most out of reusable components [BIGG89].

B. Reusable Components

Software reuse is much more than the reuse of code components. All resources or components resulting from the various stages of the software development process have the capability of being reused. These components include: [CARDS93c]

1. Domain Model

   A domain model defines the functions, objects, data, and relationships in a domain. The domain model identifies the generic requirements, represents the formal definition of the domain, and provides the general rules and principles for operating within the domain. It indicates the boundaries of the domain, the primary inputs and outputs and the standard vocabulary used. It can be used to communicate desired system features between the user and the developer of a system or group of systems.

2. Software Architecture

   A software architecture represents solutions to problems in the domain. It becomes the basis for constructing applications and mapping requirements from the domain model to design components. A generic architecture defines the basic software components, their interfaces, and the means of controlling the execution of the software. It provides a high-level generic design for a family of related applications intended to be reused to meet requirements within the domain. The generic design eliminates the need to develop a high-level design for each application within the domain. As a result, developers use these representations as specifications for reusable components.

3. Product Design

A product design, which is derived from the specification of the architecture, describes the relationship between the domain model and the work products; it is used to develop reusable components and build systems from such components.

4. Implementation Components

   Implementation components are at the lowest level and consist of: code, test information (plans, procedures and results), system/software documentation, process documentation, and generated components. In reusing code, actual code is taken from one application and reused "as is" or modified for use in another application regardless of the system design. It includes both source code (in-house or Government owned) and executable code (Commercial-Off-The-Shelf (COTS) or Government-Off-The-Shelf (GOTS)).

C. Ad Hoc, Opportunistic, and Systematic Reuse

There are different methods of performing software reuse. Developers perform reuse based on the methods and resources they have available. Students will learn how reuse is categorized based on established procedures.

1. Ad Hoc

   When there are no defined methods for performing reuse, if developers practice reuse at all, they practice ad hoc reuse. This is informal, superficial reuse [WART92].

2. Opportunistic

   The goal of opportunistic reuse is to leverage existing software components. It is up to the software developer to identify where reuse is possible, to locate components that fit the needs, and to obtain and integrate them [WART92].

3. Systematic

   The goal of systematic reuse is to leverage future software efforts by devoting time up-front to creating a suitable process. Knowledge of how and when to reuse software components within a domain is incorporated into the process. The results of which are domain-specific reusable components [PRIETO91a].

D. Small Scale Reuse vs. Large Scale Reuse

Reuse can be categorized based on the scale it is practiced on. The instructor will introduce the students to the concepts of small scale reuse and large scale reuse.

1. Small scale reuse is the reapplication of code: subroutines, object libraries, or Ada packages. Developing systems reusing only small code components does

not help reduce the large amount of work required to develop the higher level components such as the system architecture [BIGG89].

2. The reapplication of high-level components (e.g., requirements, architectures, designs) can be viewed as large scale reuse. More significant results can be realized when large scale reuse is practiced since reuse is introduced into the development process early in the life-cycle, paving the way for additional reuse. For example, if an existing set of requirements are reused, it is likely that the associated architecture and many of the designs, code, tests, and documentation components can also be reused [BIGG89].

E. Potential Risks Associated with Software Reuse

There are risks as well as benefits associated with software reuse. The instructor will discuss some of the potential risks to be considered.

1. Developing components for reuse can increase up-front development costs and time. Additional requirements may be placed on a component, such as higher levels of maintainability, portability, and reliability [HOOP89], [BIGG89], [BOEHM92].

2. The cost of developing a component for reuse may not be justified if the developing organization uses the component only once [NATO].

3. The developer must consider the cost of locating, adapting, and integrating the validated component, versus developing it from scratch [NATO], [BIGG89].

4. The reuse of software developed elsewhere imposes additional configuration management requirements if feedback on modifications and errors are provided to the supplier of the component.

5. The integrity and quality of a reusable component may not be of the standards required for the system being implemented.

6. Legacy software may not be structured to facilitate reuse. The modification of legacy software for reuse may not be feasible due to a lack of documentation, may be cost prohibitive, or may result in a component that is of lower quality than the original component [YOUR92], [BOEHM92].

7. Reuse may have a negative impact on maintenance if "as is" (e.g., Commercial-Off-The-Shelf (COTS)) components are reused [HISS92]:

   a. System complexity may increase due to excessive functionality provided by the component,

b. The component may not fit the generic architecture and therefore "stress" it,

c. Component upgrades may be significantly different than the component used to build the system.

F. Expected Benefits of Software Reuse

There are many potential benefits of software reuse. The instructor will present some of the most significant benefits.

1. Productivity Improvement

a. Improvements in productivity can be realized in all software development activities (e.g., analysis, design, coding, testing, and maintenance). Developers can produce more in a shorter period of time if they are not required to design, develop, and test every component in its entirety [NATO].

b. Reuse can significantly reduce the level of effort required during maintenance activities. Proven components can be expected to have fewer defects [SPC92].

c. Conformance to standard design paradigms reduces training requirements, thus reducing risks to schedule [NATO].

2. Increased Quality and Reliability

a. In a mature reuse program, reusable components must meet certain standards. Reusable components are certified and validated by component maintainers. Using components that meet set standards improves the overall quality of a system [YOUR92].

b. Reusable components have been proven in practice and therefore provide the potential for improved performance and reliability.

3. Improved Competitive Edge [SPC92]

a. Using reusable components can improve an organization's ability to quickly put together bids.

b. Reuse can improve an organization's ability to create realistic proposals and to outbid competitors.

c. Use of reusable components can increase an organization's ability to select alternative designs to minimize cost, time, and effort, or any other factor critical to a contract.

d. Time to market may be reduced.

A.5.3 UNIT 3 - DOMAIN-SPECIFIC REUSE

A.5.3.1 Learning Objectives

The objectives of this lecture are to provide the students with an understanding of domain-specific reuse. Students will learn about domain engineering, the activities it encompasses, and the products that result. They will learn how to perform the process of application engineering using the products of domain engineering. Upon completion of this lecture students will be able to:

1. Identify five organizations that have implemented successful domain-specific reuse programs,
2. Define domain,
3. List the activities associated with domain engineering,
4. Define domain analysis,
5. Define generic architecture,
6. Explain how application engineering uses the products of domain engineering.

A.5.3.2 Lecture Content Area

Domain-specific reuse, which utilizes a generic architecture, is a key aspect of the DoD Software Reuse Vision and Strategy. It is believed that domain-specific reuse can result in greater savings than general-purpose reuse [DOD92]. A domain is an area of activity or knowledge containing applications which share a set of common capabilities and data. Domains can be defined as vertical or horizontal. A vertical domain is a specific class of system, such as command and control or weapon systems. A horizontal domain consists of general software functions that are applicable across multiple vertical domains, such as user interfaces, mathematical programs, and graphics packages [CARDS93c]. As a domain matures, so matures the body of knowledge about it and experience in it. A mature domain has a larger number of existing systems and domain experts from which information can be drawn. Components which have been maintained and refined as the domain matures will become more reliable and effective.

A. A Historical Perspective

In response to the growing costs of software development, organizations have realized a need to refine their development processes. The following organizations chose domain-specific strategies as the means to achieving a reduction in the costs associated with development. Students will gain an understanding of actual savings being realized by existing domain-specific reuse projects.

1. Common Ada Missile Packages (CAMP):

   CAMP, an early DoD reuse effort, was one of the first domain-specific reuse projects. Its goal was to establish the feasibility and value of reusable Ada software within the Missile Operational Flight Software domain. The project began in 1984. Currently there are over 500 CAMP components. The CAMP project was the first explicitly reported domain analysis experience. They acknowledge that domain analysis is essential and that it is the most difficult part of establishing a software reusability program. The CAMP project has played an important role in technology transfer [FRAKES92], [PRIETO90].

2. Restructured Naval Tactical Data Systems (RNTDS):

   The RNTDS project began at the Fleet Combat Direction Systems Support Activity (with support from Paramax Systems Corporation) in 1976. The goal of the project was to reduce costs and time associated with development and maintenance of systems with similar functionality. One performance specification was developed for the RNTDS domain. An initial program baseline was derived for the main class of ships developed. Requirements for classes which differed from the baseline were then defined as deltas, rather than as new sets of requirements. A library was established to maintain reusable parts. Reuse potential across RNTDS averages close to 90%. There exists 77% commonality across programs. The use of RNTDS has resulted in 26% fewer required labor hours [GOOD92].

3. Raytheon Missile Systems Division:

   This successful reuse experience in business applications reported up to 60% reuse resulting in a 50% increase in productivity. Raytheon recognized the redundancy in its business application systems and instituted a reuse program. They analyzed their applications and identified three common classes. Standard architectures were developed for each class and a library was developed and populated for future development efforts [NATO].

4. Fujitsu's Software Development for Electronic Switching Systems (SDESS):

   Fujitsu analyzed its electronic switching systems, developed a library and filled it with reusable components. The library is staffed by domain experts, software engineers, and reuse experts. Use of the library is mandatory. All design review teams include members of the library staff. Fujitsu has experienced a significant improvement in the number of projects that are completed on schedule. Before the program, 20% of electronic switching systems were delivered on schedule. Once the project was instituted, 70% are delivered on schedule [PRIETO91a].

5. Nippon Electronics Corporation (NEC) Software Engineering Laboratory:

   NEC analyzed its business applications and identified 32 common designs and 130 common algorithms. A library was created to house the common components and was integrated into NEC's software development environment which enforces reuse. NEC has reported a 6.7 fold productivity improvement and a 2.8 fold quality improvement [NATO].

B. Domain-Specific Processes

This section is intended to provide students with an understanding of the processes involved in domain-specific reuse. Students will learn the inputs and outputs of domain engineering and how they feed into application engineering activities. The instructor will explain how these activities map to common software development models, such as DOD-STD-2167A.

1. Domain Engineering

   The instructor will explain how domain engineering is the basis for reuse-based software engineering. Domain engineering results in the organization of domain knowledge for future software development use. Domain engineering is performed for a family of systems. The resulting products (i.e., domain model, generic architecture, product design, and implementation components) are used to develop applications for various projects.

   a. Domain Analysis

      The process of identifying, collecting, organizing, analyzing, and representing the relevant information in a domain based on the study of existing systems and their development histories; knowledge captured from domain experts; underlying theory; and emerging technology within the domain [DISA92b], [PRIETO90].

   b. Architecture Development

      Using the domain knowledge gathered during a domain analysis, domain engineers develop a generic architecture. A generic architecture is the high-level paradigms and constraints that characterize the commonalities and variances of interactions and relationships among components in a system. The goal is to develop architectures which support the creation of systems that can accommodate change [MITRE92].

c. Creating Reusable Components

Using the generic architecture, domain engineers create reusable components which are catalogued into a reuse library for use by application engineers.

d. Component Recovery

If legacy systems are to be included as reusable components in the library, domain engineers may need to modify them to meet constraints of the generic architecture. Often this is accomplished by creating a wrapper around the original software [YOUR92], [CARDS92a].

e. Component Management

The DoD Software Reuse Vision and Strategy document defines the concept of centrally managed reuse within a domain [DOD92]. It has been recommended that this be done via a domain management office. A domain management office will perform the domain engineering tasks described above (a through d). In addition, the domain management office assists application engineers in refining requirements and defining system architectures. Results of application engineering and development (e.g., designs, specifications, code, and documentation) are provided to the domain management office, both incrementally during development and in their entirety upon completion. This assists the domain management office in updating, maintaining, and managing the domain model and associated domain knowledge [CARDS93c].

2. Application Engineering

The instructor will explain how application engineering is performed using the products of domain engineering [CARDS93a]. The students will gain an understanding of the importance of identifying reuse opportunities and providing feedback to the domain engineering process, through domain management, during each development activity.

a. Requirements Analysis

Reusing requirements and related components during requirements analysis activities can reduce risks. Reuse libraries and components can provide the customer and the developer with a common basis of understanding if the same domain engineering products are used for system specification and system development. Deriving or reusing requirements and capabilities from existing systems enhances reliability, lowers program risks, and strengthens customer confidence. Additionally, the cost and schedule of building a system may be reduced. Mapping customer requirements to existing system requirements may categorize some of the customer requirements as implementation constraints. The generic architecture and domain requirements may be used to propose equivalent but more generic requirements. This process may eventually result in more portable and reusable systems.

Students will learn that the systematic reuse of requirements, designs, and code over multiple systems can reduce the level of effort necessary for the requirements analysis task and result in cost and schedule savings. Students will learn how domain-specific reuse facilitates requirements analysis activities by supporting prototyping.

b. Design and Development

Reusing existing designs and related components during design and development activities can reduce risks to the project, and the introduction of errors during the development of new designs and code can be reduced. Reused design and code components may not be bug free but they have been used, refined, and tested so they are usually of higher quality than new components.

Large cost and schedule impacts of designing and coding from scratch can be reduced. Lessons learned from previous design experience in a domain can be exploited. High personnel turnover makes this an important issue, but systematic reuse of designs takes advantage of this previous experience. Reuse reduces the likelihood of being stuck with a "closed" system that is difficult to port or evolve, which can happen if generic architectures are ignored.

c. Integration and Testing

While integration testing is still necessary, reusable components that have been qualified for a reuse library do not have to be retested individually if they have not been modified and meet the certification standards of the project. If the development process is architecture-centric, component interfaces will be well defined and integration will be straight forward. Developers should take advantage of test materials supplied by the library for unit and integration testing.

d. Maintenance

Reusing domain models, generic architectures, product designs, and implementation components developed for reuse should result in systems that are easier to maintain. "The desirable characteristics of reusable assets are much the same as those of maintainable assets" [STARS92a]. In addition, the reuse of existing components which are supported by their source can greatly reduce the level of effort required for maintenance activities, since modifications will be made for the developer.

A.5.4 UNIT 4 - DOMAIN ANALYSIS OVERVIEW

A.5.4.1 Learning Objectives

The objectives of this lecture are to provide students with an understanding of domain analysis and to build confidence in the products produced by the domain analysis process. This unit is not intended for students who will be performing domain analysis, but for students who will use domain analysis products. This lesson will be complete enough to instill an awareness of domain analysis techniques and their role in systematic software reuse. Upon completion of this lecture the student will be able to:

1. Identify several domain analysis methodologies and describe what they have in common,
2. Explain how a domain should be assessed for domain analysis,
3. Explain the importance of determining the boundaries (scoping) of the domain being analyzed,
4. Define context diagram,
5. List the three main questions that should be asked in performing context analysis,
6. Explain what information is captured by domain modeling,
7. List and explain the products of domain analysis,
8. Describe how a generic architecture is derived.

A.5.4.2 Lecture Content Area

The CARDS Program is not prescribing one particular approach to domain analysis, but references [COHEN92], [DISA92b], [HOLI89], [PRIETO90], [TRACZ92] and [WART92] provide a solid foundation for exposition and course activities. These papers have been used to develop this section of the course. It is assumed that the students have a working knowledge of structured analysis and object-oriented analysis techniques.

A. Domain Analysis Methodologies

There are many domain analysis methods [HOLI89]. Each method emphasizes different aspects of the domain analysis process, but they all describe a similar process. The basic steps in this process include:

- Scope or bound the domain,
- Collect and organize domain information,
- Define and classify domain specific terms,
- Create a domain model,
- Define a generic architecture for the domain.

Most domain modeling methodologies formulate similar models and viewpoints of the domain. These models often include feature models, functional models, object models, and composition rules. Some of the common modeling techniques used are entity-relationship models, state diagrams, hierarchical models and other structured analysis and design techniques. The domain model captures the common parts of the domain along with the differences. The data commonality and the control flow for the functionality is captured in a functional model. This functional model is then used to generate the requirements for a generic architecture and the reusable components that implement the architecture [TRACZ92]. Students will be provided with an overview of several different domain modeling approaches used by different domain analysis methods.

1. Prieto-Diaz Domain Analysis Model [PRIETO91b]

   The Prieto-Diaz Domain Analysis Model is based on a methodology for deriving specialized classification schemes. This method uses concepts from library science, specifically literary warrant, to develop generic characterizations and standard models. Domain terms are identified and facets are defined. Facets are ordered by relevance and domain terms are ordered within the facets. A semantic net is produced containing frames, representing domain concepts and rules defining their relationships.

2. DISA Domain Analysis Guidelines [DISA92b]

   The DISA method is an object-oriented approach that produces class diagrams and assembly (whole-part) diagrams to model domain structures and their relationships. The whole-part diagrams define the composition of the domain, while the class diagrams describe the variations in objects. Models are generalized and made implementation-independent.

3. Joint Integrated Avionics Working Group (JIAWG) Object-Oriented Domain Analysis (JODA) [HOLI89]

   The JODA method is an object-oriented approach that produces scenarios. Scenarios relate special cases to the normal case. They describe the major threads of processing from a stimulus entering the system until it completes processing. Scenarios support the utilization of the domain model and components by relating external events to the services of the domain.

4. Feature Oriented Domain Analysis (FODA) [COHEN92]

   The FODA method produces feature models. Feature models provide the end user's perspective of the capabilities (mandatory, optional, and alternative) of applications in a domain. Compatibility between features are defined as composition rules.

B. Domain Identification

It is necessary to choose the right domains to analyze. A domain must be stable and well understood to be a good candidate for domain analysis [TRACZ92]. Students will gain an understanding of how to assess whether or not a domain is a good candidate for domain analysis. A discussion of how a domain is chosen will occur and an illustration will be provided by the instructor. Some questions to consider in making domain selections are:

1. Is the domain broad or narrow?

2. Is the domain mature and well understood?

3. Is the domain stable or changing continuously?

4. Is the domain strongly dependent on technology?

5. Is the domain based on well established principles, methods, and formalisms?

The instructor will discuss the importance of identifying the resources that can be drawn upon in performing domain analysis.

1. Who do they have to work with?

2. What do they have to work with?

3. How will they verify their models?

C. Defining the Domain and Domain Modeling

In performing domain analysis, the boundaries of the domain must be properly scoped. This process places the domain relative to other domains and defines the external entities and data flows between the external entities and the domain. Students will learn the importance of determining the boundaries of the domain being analyzed and receive hands-on experience developing models.

Activity: The students will create a context model and a feature-oriented domain model for the domain being analyzed. These will not be independent exercises. The instructor will lecture, allow for discussion and then the students will perform exercises. This will continue throughout the activity. Students will work in groups on the exercises. The instructor will provide the name of the domain being modeled, a short description of the domain and the general user needs to be satisfied by applications in this domain. The instructor, assuming the role of a domain expert, will validate the models produced by the students.

1. Context Diagram

   The students will draw a high-level block diagram showing the context of the domain and the relationships between the entities in the domain (e.g., an entity-relationship diagram). The instructor will discuss the context of the domain and the domain itself and then let each group develop their own model. A time limit will be set on this exercise. The instructor will collect the models developed and evaluate them at a later time. Models developed by the instructor will be handed out when the students' models are collected. The following types of questions will be provided to the students to guide them through this activity:

   a. What is inside the domain?

      What are the classes of applications in this domain?

      What primary functions/objects/things are in the domain?

      What kinds of trade-offs exist in this domain?

   b. What is outside the domain?

      What functions/objects/things in the domain are outside the scope of the sub-domain chosen to be analyzed?

      What are similar/related domains and sub-domains?

      How does this domain relate to other domains?

   c. What is on the borders of the domain (input/output)?

      What are the inputs to the domain?

      What are the outputs from the domain?

Where do inputs to the domain come from?

Where do outputs from the domain go to?

2. Domain Model

The students will develop a feature-oriented domain model illustrating the end user's perspective of the capabilities of applications in the domain [COHEN92]. The instructor, acting as the customer, will discuss some of the perceived capabilities of applications in the domain. The instructor will set a time limit on this exercise. The instructor will collect the models developed and evaluate them at a later time. Models developed by the instructor will be handed out when the students' models are collected. The following types of questions will be provided to the students to guide them through this activity:

a. What capabilities does the user believe domain applications have?

b. Are there capabilities not yet defined that may be required in the future?

c. What relationships exist between capabilities identified?

d. If one capability exists, which other capabilities must also exist?

e. If one capability exists, which other capabilities may not exist?

3. Domain Analysis Products

The instructor will provide the students with domain models, a dictionary of the domain-specific terminology, a "generic" requirements specification document, and a generic architecture. The instructor will review the domain analysis products with the students. The instructor will point out how the generic architecture emphasizes the "standard" concepts in the domain and how it represents many different applications in the domain. Some generic architectures provide interface specifications among the components of the domain model.

The instructor will discuss how the generic architecture was developed and how general design and implementation constraints on the architecture were determined and applied to a domain model. In developing the generic architecture software constraints, hardware constraints, performance constraints and general design constraints (e.g., fault tolerance, security, and safety) are all considered. The students must understand the structure of the generic architecture in order to use it.

A.5.5 UNIT 5 - SOFTWARE REUSE LIBRARIES

A.5.5.1 Learning Objectives

The objectives of this lecture are to introduce the students to reuse libraries, library mechanisms and certification and qualification techniques. Students will receive hands-on experience using a reuse library. Upon completion of this lecture the student will be able to:

1. Explain why the reuse library is a tool for reuse and not a solution,
2. Define component-based library,
3. Define model-based library,
4. List three classification methods and describe them,
5. Describe when a particular form of search and retrieval is beneficial,
6. Explain component certification,
7. Explain component qualification,
8. Define common metrics,
9. Define domain metrics,
10. Explain how system composition tools work,
11. Explain how system generation tools work,
12. Navigate through a reuse library.

A.5.5.2 Lecture Content Area

A. The Library - A Reuse Tool

Software reuse libraries are a tool for reuse; they are not the reuse solution. The instructor will discuss how most individuals maintain their own pool of reusable components and how compilers are equipped with their own set of reusable libraries. The students will gain an understanding of how the software reuse library concept is now being extended to include publicly accessible libraries of life-cycle artifacts. This extension is essential if the full potential of software reuse is to be realized. The greatest reuse savings can be seen when a component is reused many times, across projects. organizations may benefit by making their own components available internally to all projects and by acquiring components available externally [NATO].

B. Types of Libraries

Reuse libraries can be classified by the methods and techniques used to organize components. A library is either component-based or model-based [WAL92]. Students will learn how libraries of these two classes are similar and how they differ.

1. Component-Based Libraries

Component-based libraries are similar to book libraries. They can be thought of as software warehouses. Component-based libraries are frequently general-purpose reuse repositories. However, the component-based approach can be used effectively to support domain-specific reuse as well. As the name suggests, the central focus of a component-based library is the component. Library users search for individual reusable components. Library support focuses on developing efficient and effective search and retrieval mechanisms.

2. Model-Based Libraries

Model-based libraries are organized around the principle that what matters in a repository is the context in which reusable software components are used and the relationships among components. As the name suggests, the focus of a model-based library is the model. Models can include models of requirements, architectures, design decision and rationales, and the software which "implements" these models. Encoded in the model is the organization's "corporate memory" of the domain. The model-based library can facilitate the variant composition of subsystems tailored to application-specific requirements.

C. Library Mechanisms

There are many different types of library classification, search, and retrieval methods. Students will gain an understanding of some of the common methods and how they differ.

1. Classification Methods

"Classification is grouping like things together. All members of a group, or class, produced by classification share at least one characteristic that members of other classes do not. Classification displays the relationships among things and classes of things. The result is a network or structure of relationships" [BIGG89]. The instructor will develop a lecture that presents a brief overview of various classification methods (e.g., Dewey decimal, semantics, faceted, indexed, and keywords) [STARS92b], [PRIETO91b], [FRAKES90].

2. Search and Retrieval Methods

   The instructor will provide the students with an overview of various search and retrieval methods (e.g., full text searching, facet searching, keyword searching, knowledge-based searching, and browsing). Students will learn what the benefits of different methods are for different environments.

D. Library Component Certification and Qualification

The largest resistance to reuse comes from developers who do not trust reusable components. Developers must be able to trust that components have been properly developed and that using a previously developed component does not add significant risks to a project and may reduce or eliminate many risks. Library component certification and qualification efforts are being developed to ensure developers that components can be trusted. Component certification is the process of determining if a component being considered for inclusion in the library meets the requirements of the library and passes all testing procedures. Evaluation takes place against a common set of criteria [ASSET92], [RAPID91]. Component qualification is the process of determining if a potential component meets all of the requirements and constraints of a domain's generic architecture. Evaluation takes place against domain criteria [CARDS02f], [PRISM92a].

Both common metrics and domain metrics are considered in evaluating components for inclusion in a domain-specific reuse library. The common and domain metrics evaluations will lead to a recommendation from the evaluators as to whether or not a component should be accepted for inclusion in the library. Students will gain an understanding of how reusable components are evaluated for inclusion in a domain-specific reuse library.

1. Common Metrics

   Common metrics are criteria used to evaluate components regardless of domain. Example criteria for common metrics are reliability, maintainability, portability, and security.

2. Domain Metrics

   Domain metrics are criteria used to evaluate components based on domain requirements, architectural constraints, and implementation constraints. Example criteria for domain metrics are form, fit and function [PRISM92a]:

   a. Form

      The conformance of a product to the component interface definition, as defined by the generic architecture.

   b. Fit

The conformance of a product to performance constraints, fault tolerance requirements, and security requirements, as defined by the generic architecture.

c. Function

The conformance of a product to the component's functional capabilities, as defined by the generic architecture.

E. Library Supported Tools

Reuse libraries may provide tools to assist the application engineer in developing systems, subsystems, and prototypes. These tools can be categorized as either composition-based or generation-based. Students will gain an understanding of how these tools operate.

1. System Composition Tools

System composition tools can be used by application engineers to construct new applications from previously developed or newly generated parts. This is typically done by identifying, understanding, evaluating, and selecting appropriate components and integrating them to meet specific system needs. System composition tools are supported by domain models which provide reusable components, a generic architecture and embedded domain knowledge [STARS92c].

2. System Generation Tools

System generation tools can be used by application engineers to develop new applications or subsystems through the use of parameterization or specification languages. A generation tool accepts an application's specifications and generates components for the target system [STARS92c].

F. Using a Domain-Specific Library

Activity: The instructor will present a domain-specific software reuse library which corresponds to activity exercises of the other units. The instructor will provide the students with an easy to follow reference sheet for accessing and using the library. The instructor will review this in detail with the students. Students will learn to navigate through the library.

A.5.6 UNIT 6 - APPLICATION ENGINEERING

A.5.6.1 Learning Objectives

The objectives of this lecture are to provide students with a hands-on experience performing application engineering using the products of a domain engineering effort and a reuse library. The instructor will guide the students through application engineering activities. During each activity, the instructor will emphasize considerations students must make in light of reuse and discuss how the activity maps to common software development models (e.g., DOD-STD-2167A). This unit assumes that domain engineering has been performed for the domain to be used for the activities and that domain engineering products exist. The instructor will have the students consider different options available in the event certain domain engineering products or tools are not available. The CARDS Engineer's Handbook [CARDS93a] should be used for guidance through these activities. Upon completion of this lecture the student will be able to:

1. Define prototyping,
2. Explain how the domain-specific library and generic architecture can be used in defining and clarifying application requirements,
3. Describe how the generic architecture is instantiated,
4. Explain how an architecture-centric reuse process facilitates integration,
5. Explain why testing efforts may be greatly reduced using proven, unmodified, reusable components,
6. Explain why maintenance efforts may be greatly reduced using the products of domain engineering.

A.5.6.2 Lecture Content

This section will use instructor-provided domain models, system requirements and a domain-specific library to illustrate and support the development of a system architecture and a system prototype. This is an intensive hands-on part of the course.

A. Instantiation of the Generic Architecture

Activity: The instructor will provide the students with domain models, system requirements for the application to be developed, a generic architecture, and a domain-specific software reuse library. The student will define a specific application architecture. This activity will illustrate how domain engineering products can be used to define and clarify application requirements. The instructor will act as an expert in the domain being modeled and as the customer for the application being developed. Using the generic architecture provided by the instructor, students will use the domain-specific software reuse library to further define and clarify the customer requirements. The students will use the browsing mechanism provided by the library to search for information. The instructor will explain how this activity differs depending on the type of library used (component-based or model-based), and discuss how this activity maps to development phases of common models, such as DOD-STD-2167A's requirements analysis phase.

1. Explore the Domain-Specific Software Reuse Library

   a. What is the application being defined in the requirements?

   b. How will the components contained within the software reuse library help shape the application design and decision making?

   c. Which components will be likely candidates for reuse?

2. Clarify the Requirements

   Using the software reuse library, the students and the customer (instructor) will clarify ambiguous requirements and isolate differences between the requirements and the generic architecture.

   a. Do the requirements correspond to the generic architecture?

   b. Are there features or functions specified in the requirements that are not represented in the generic architecture?

   c. Are there features or functions in the generic architecture that have been left out of the requirements that must be included?

   d. Have all constraints on the system been adequately specified in the requirements?

   e. Do the requirements include conflicts between features (mutual exclusion) which have not been clearly identified?

   f. Do the requirements include relationships between features which have not been clearly identified?

   g. Does development experience exist for a component or project which would suggest the requirements be modified in some way?

3. Revise the Requirements

   The students and the instructor will create a new set of requirements by mapping all of the issues identified above onto the original requirements.

4. Instantiate the Generic Architecture

   a. For each set of alternative features or functions, choose those which correspond to the application requirements.

b. For each optional feature or function, decide if it will be included in the application's archi[illegible]e.

c. What features do not exist in the generic architecture? (Note: These are the components which do not exist in the software reuse library and must be developed or obtained elsewhere.)

B. Creating a Prototype

The software components contained within the software reuse library have been designed in a manner that enables them to be reused without detailed knowledge of the code itself. These components can be assembled to create a prototype. Prototyping is the acquisition of software [illegible]ct knowledge through disciplined experimentation. This includes both exploratory prototyping for information gathering, and production prototyping, where the prototype is continuously improved until it becomes the final product [CARLS90]. The instructor will discuss how different reuse tools facilitate prototyping and identify different phases of common development models in which prototyping can be exploited (e.g., DOD-STD-2167A's requirements analysis, design, and implementation activities).

Activity: The students will create a prototype using the software reuse library, the revised requirements and the application architecture defined in the previous section. Students will produce prototypes which compile and run using available reuse tools (e.g., system composition tools, system generators).

1. Collect Components

Using the software reuse library the students will collect components.

a. What are the system requirements?

b. What are the constraints on the system?

c. Are composition constraints recognized that were not identified earlier?

2. Record Issues, Trade-offs, and Design Rationale.

In cases where the students have to choose between several reusable components, the student will provide a rationale for why one was chosen over another.

a. What were the trade-offs?

b. Did this choice drive future decisions?

c. Did past decisions influence this choice?

3. Link Artifacts

   The students will link the components collected from the software reuse library and present a brief demonstration of the prototype to the instructor. The instructor, acting as the customer, will evaluate the prototype and the associated component choice rationale. The instructor will discuss these with the students.

   a. Are there problems associated with interfacing any two components taken from the library?

   b. What input must be supplied?

   c. How much of the system was prototyped?

   d. Was it necessary to relax requirement constraints to create the prototype?

4. Integration Testing

   The instructor will discuss integration testing with the students. The instructor will explain that it may not be necessary to test individual library components that have not been modified.

   a. Does the prototype function as required?

   b. Does the prototype perform at required levels?

   c. What parameters need to be modified to meet system performance requirements?

5. Prototype to Actual System

   a. What components need to be developed and integrated?

   b. What changes to the prototype may the customer request?

6. Component Adaptation

   a. What components require adaptation?

   b. Are the required changes small?

c. Will it be cost effective to adapt the component instead of developing it from scratch?

d. Will the development of an interface wrapper fill the need?

7. Maintenance

a. How will the domain engineering effort help to reduce potential maintenance requirements?

b. Which components have a greater probability of requiring maintenance?

c. Are anticipated maintenance requirements high or low for the entire system?

d. How do anticipated maintenance requirements compare to maintenance requirements for a system developed without reusing proven components?

A.5.7 UNIT 7 - REUSE BARRIERS

A.5.7.1 Learning Objectives

The objectives of this lecture are to provide students with an understanding of the barriers they may encounter in attempting to incorporate reuse into their current development process and with suggestions on how to overcome the barriers. Upon completion of this lecture, the student will be able to:

1. List actual as well as potential barriers to reuse,

2. Identify current actions underway to address and resolve barriers,

3. Offer solutions to actual and potential barriers, and an approach to accomplish resolution.

A.5.7.2 Lecture Content Area

Reuse calls for a significant change in software development practices. The shift to software engineering processes which incorporate reuse principles must be addressed at multiple levels from management through technical personnel. Reuse will not be successful until it becomes an inherent part of the software development process. Barriers to reuse exist at all levels. However, more solutions are being recognized as reuse concepts and processes mature.

A. Barriers to Reuse

Barriers to reuse can be categorized into three fundamental groups. This categorization can sometimes be ambiguous. Not all barriers to reuse fit exactly into one category. Students will gain an understanding of the different types of barriers they may encounter.

1. Technical

   Technical barriers may affect the actual development of software. An example of a technical barrier is the nonavailability of a component.

2. Programmatic or Non-technical

   Programmatic, or non-technical, barriers are sometimes loosely referred to as "management issues." Examples of programmatic barriers to reuse include a lack of cost model availability, and lack of education in reuse strategies and techniques which impacts developers' ability to incorporate reuse into software development.

3. Cultural

   Cultural barriers exist due to the internal environment and the psychodynamic and demographic characteristics of the system and software engineers themselves. An example of a cultural barrier is the lack of trust developers may have for components they did not create. System and software engineers will be confronted primarily with technical barriers to reuse. However, programmatic and cultural barriers can sometimes influence the sphere of operations and the attitudes of colleagues in the environment in which engineers must work and are therefore presented in this unit.

   For system and software engineers, overcoming barriers to reuse can be difficult. For example, a skilled engineer may know what component is needed, but may not know where to find it. In this case, a solution would involve gaining access to one or more reuse libraries. There is not a solution to every technical barrier to reuse; however, many barriers do have possible solutions. The types of barriers an engineer will encounter depend upon the engineer, the organization, and available technology. Following are some examples of barriers an engineer may encounter. Additional examples can be found in [CARDS93a], [CARDS93c], [SPC92], [YOUR92], and [HOOP89].

B. Technical Barriers

1. The software reuse library may not depict the current state of the domain. The reuse library must be constantly updated with feedback from projects where an application was built using new technology. Engineers should keep abreast of latest technology and their implications/ramifications on applications within the domain.

2. There may be inconsistencies and ambiguities in terminology and how domain components are classified in the library. Domain management offices will help to ensure the use of glossaries, formal languages and models, and domain dictionaries.

3. Knowledge representations for generic architectures may not be adequate. Developers must ensure that the architectures chosen meet current and projected user requirements.

4. Using reusable code components (GOTS and COTS) may introduce risks if they have not been developed and tested with the new application in mind, and if they contain system and compiler dependencies. Careful evaluation of components for the target system, extra unit testing, and thorough integration testing can reduce this risk.

5. Use of COTS and GOTS components can increase risk due to cost of license and maintenance agreements, the potential that support for the component may be dropped, and the possibility that the component may provide more than needed. These factors must be weighed in choosing components to fill an architecture.

6. Improper integration of reusable software components can increase risks. Reuse of an existing generic architecture which identifies component interfaces can help ensure that components are chosen and created to properly integrate.

7. A reusable component may not easily adapt to the new application environment. However, application environment simulation can be used to verify that the component will adapt to the new application.

8. Domain engineering products may be difficult to locate. The expansion of, and interoperability between, existing libraries such as CARDS, DSRS, ASSET, etc., will help overcome this barrier. There exist several centers which distribute information on available reuse programs, including ASSET, the Ada Joint Program Office (AJPO), and the National Technology Transfer Center (NTTC). If domain engineering has not been performed for the organization's product line, the organization should consider instituting a domain engineering program for the domain in question. The program may include the establishment and maintenance of a reuse library.

C. Programmatic Barriers

1. The cost of employing reuse may be prohibitive. However, the Government is taking the initiative to identify domains for which employing reuse will pay

long-term dividends. The Government is setting up the necessary infrastructure to enable successful reuse in building future systems in these domains.

2. Reuse activities may conflict with the organization's standard development process and methodology. Reuse must be made an inherent part of the software development process. Engineers will not be successful trying to practice reuse in isolation.

3. Reuse activity may be low because engineers are not trained in utilizing reusable components or recognizing opportunities to apply them. Part of the reuse infrastructure the Government is developing focuses on providing training courses, such as this one, to teach engineers how to incorporate reuse into their current software development processes.

4. Management may not become involved or provide support for reuse. Continued education in the potential benefits of reuse (cost and schedule savings, productivity and quality improvements), increased collection and analysis of metrics illustrating the benefits of reuse, and a "phase-in" approach to reuse will help generate increased involvement and support. Management must "buy into" reuse. Effective communication must be instituted, and a flow and exchange of ideas and information in both directions must be continually fostered.

D. Cultural Barriers

Engineers may not trust components developed for reuse. They may believe that they can produce components that are better. They may believe that reusing components will remove the creative element from their work. Management must determine what kinds of cultural problems can be anticipated and why. They must understand what kinds of rewards are effective motivators, and how best to institute implementation of reuse. In the long-term, increased knowledge and experience of personnel will help overcome cultural bias. Continued maturity of reuse in the engineering/development process together with the continued development of reusability standards, libraries, and tools will build trust in the methods. Reuse successes and positive user feedback will instill confidence in the processes and products produced.

A.6 COURSE IMPLEMENTATION QUESTIONS AND ANSWERS

1. What is a reasonable size for the activity projects?

Due to the limited time of the course, small projects will be presented. The project should not be an example from the domain in which the students work. This will help to keep the students focused on the methods being presented and not on the example itself. The examples will be small and yet large enough to expose the students to all the concepts and methods they need to incorporate software reuse into their software development processes.

2. What is the role of the instructor in the course?

   The instructor must provide project description, assistance in assignments, guidance during project executions, lectures on software reuse, software reuse libraries, domain analysis, generic architecture development, application architecture development, reuse tools, and perform activity evaluations. Instructor intervention may be needed to direct students in the right direction during activity exercises. To minimize the risk of student failure during the activity exercises, it is necessary for the instructor to be very comfortable with the exercises being presented and their solutions. The instructor will be prepared for students to encounter problems.

3. How are the activity exercises evaluated?

   Upon completion of each activity, the instructor will review the work of the students. The activities will not be graded. The instructor will provide the students with comments, where appropriate, indicating the areas they need to give more attention to and those they did well. Upon returning the activities to the students, the instructor will also provide the students with a "solution." The instructor will discuss the activity exercise with the students before moving on to the next activity to ensure that all students are relatively comfortable with the methods presented in the previous activity. Each activity builds upon the products of the previous activity.

4. How can the instructor ensure all students get off to a fast start on the activity exercises?

   Preliminary preparation for the activity exercises by the instructor is critical to the overall success of the course. All materials required by the students to complete the exercises must be prepared and thoroughly reviewed by the instructor. The instructor must be certain that all required equipment is present and operational. A fast start is essential if the students are to complete the activity assignments in the given time frame. To minimize the time it takes for a student to get started on the assignment, the instructor will have all materials ready for the student on the first day of the course. The students will then have some time to consider the exercise before beginning it. The instructor will be prepared to give considerable help to all students in the critical first key steps of the activity exercises. Specifically, the instructor may choose to make some of the activities group exercises and some individual exercises. This will keep slower students from falling too far behind.

A.7 POSSIBLE ACTIVITY EXERCISES

Following is a list of possible subject domain areas for activity exercises. The size of the domain chosen for course activities should be based on the amount of time allotted for activity completion.

1. Automatic Teller Machines (ATMs)
2. Banking Systems
3. Hospital Patient Monitoring Systems
4. Text Editors (Word Processing Systems)
5. E-mail Systems
6. Calculator Programs
7. Menu Manager Systems
8. Elevator Systems
9. Walkman Devices
10. Warehouse Systems
11. Flight Reservation Systems
12. Message Processing Subsystems
13. Personnel Records Management
14. Command Centers
15. Missile Systems
16. Avionics Systems

A.8 HANDOUT - COURSE EVALUATION FORM

Please complete the following course evaluation form.

1. Name (Optional):

2. Organization (Optional):

3. I perform the following tasks as part of my job responsibilities (circle all that apply):

   Specify systems/software

   Design systems/software

   Develop systems/software

   Test systems/software

   Verify systems/software

   Control systems/software

   Manage systems/software

   Other___________

4. I will be able to apply the course content to my job duties (Y/N).

5. Before I attended this course, I was familiar with:

Table 5-1

	Novice	Knowledgable	Experienced	Expert
Structured Analysis				
Model-Based Engineering				
Domain Analysis				
Software Reuse Libraries				
Domain-Specific Software Reuse Libraries				
Generic Architecture Development				
System Engineering				
Software Engineering				

6. The length of this class was:

   Too Long

   Too Short

   Just Right

7. The material covered was:

   Too Specific

   Too General

Adequate

8. The time given for each unit was adequate: (Y/N)

   If no, indicate which units did not provide adequate time.

9. I had plenty of opportunity to participate and ask questions: (Y/N)

10. Were your questions adequately answered? (Y/N)

11. The examples helped clarify the concepts:

    Very Much

    Moderately

    Not at All

12. The course handouts were:

    Very Useful

    Moderately Useful

    Not Useful

13. I would recommend attendance to my colleagues: (Y/N)

    If no, why not?

14. I have attended other software reuse training classes or seminars: (Y/N)

    If yes, which ones?

15. I would be interested in attending other software reuse training seminars: (Y/N)

16. I would be interested in receiving additional information on software reuse: (Y/N)

17. I believe domain-specific software reuse is very valuable in the development of software systems: (Y/N)

18. The training facilities and equipment were adequate: (Y/N)

19. The instructor's knowledge of course material was:

    Poor

    Fair

    Good

Excellent

20. The instructor's ability to explain the course material was:

    Poor

    Fair

    Good

    Excellent

21. The instructor's preparation of course material was:

    Poor

    Fair

    Good

    Excellent

22. Additional Comments:

A.9 HANDOUT - GLOSSARY

Ad-Hoc Reuse -

Reuse is practiced ad-hoc when there are no defined methods for performing reuse.

Application -

A system which provides a set of general services for solving some type of user problem.

Architectural Constraints -

The definition of relationships/interfaces among components of a generic architecture.

Architecture-Centric Reuse -

Reuse is architecture-centric when the component development process and the application development processes are based on a generic architecture. The goal of an architecture-driven process is to achieve black-box reuse.

Black-box Reuse -

Black-box reuse is achieved when application engineers can compose systems by plugging together different reusable components based on an application's requirements.

Certification -

The process of determining that a component being considered for inclusion in the library meets the requirements of the library and passes all testing procedures.

Common Metrics -

Criteria used to evaluate components regardless of domain. Example criteria for common metrics are reliability, maintainability, portability, and security.

Component-Based Library -

Component-based libraries are similar to book libraries. They can be thought of as software warehouses. The central focus of a component-based library is the component.

Domain -

An area of activity or knowledge containing applications which share a set of common capabilities and data.

Domain Analysis -

The process of identifying, collecting, organizing, analyzing, and representing the relevant information in a domain based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain.

Domain Engineering -

An encompassing process which includes domain analysis and the subsequent construction of components, methods, tools, and supporting documentation that address the problems of system/subsystem development through the application of the knowledge in the domain model and software architecture.

Domain Metrics -

Criteria used to evaluate components based on domain requirements, architectural constraints, and implementation constraints.

Domain Model -

A definition of the functions, objects, data, and relationships in a domain, consisting of a concise representation of the commonalities and differences of the problems of the domain and their solutions.

Domain-Specific Reuse -

Reuse that is targeted for a specific domain (as opposed to reuse of general purpose work products). It typically involves reuse of larger workproducts (subsystems, architectures, etc.) than general purpose reuse.

Domain-Specific Software Architecture -

A software architecture that has been generalized based on actual and projected commonalities of systems in a domain. May also be referred to as a generic architecture.

Facet -

A perspective, viewpoint, or dimension of a particular domain which can be represented as a class.

Generic Architecture -

High-level paradigms and constraints that characterize the commonality and variances of the interactions and relationships between the various components in a system. May also be referred to as a domain specific software architecture.

Horizontal Domain -

The knowledge and concepts that pertain to a particular functionality of a set of software components that can be utilized across more than one application.

Implementation Constraints -

Constraints to be satisfied when components of a generic architecture are composed into specific systems.

Interoperability -

The ability of two or more systems to exchange information and to mutually use the information that has been exchanged.

Knowledge Blueprint -

A flexible plan to transition knowledge to the community.

Large Scale Reuse -

Large scale reuse is the reapplication of high-level components (e.g., requirements, architectures, designs).

Library-Assisted Reuse -

Reuse is library-assisted when there exists a library to support the application domain. There may be more than one library and they may be interconnected.

Megaprogramming -

Megaprogramming is achieved when systems and subsystems can be viewed as black-boxes that meet certain requirements. These systems can be reused in building other systems without the developer requiring detailed knowledge of the system's internal structures.

Model-Based Library -

Model-based libraries are organized around the principle that what matters in a repository is the context in which reusable software components are used and the relationships among components. The focus of a model-based library is the model (requirements, architectures, design decisions and rationales) and the software which implements these models.

Opportunistic Reuse -

Reuse is practiced opportunistically when it is up to software developers to identify when reuse is possible, locate reusable components, and integrate them.

Process-Driven Reuse-

Software reuse is process-driven when it is an integral and transparent part of both the software engineering process and the broader acquisition process.

Prototype -

A model of a system (sometimes scaled down, but accurate) used to verify the operational process prior to building a final system.

Qualification -

The process of determining that a potential component is appropriate to the library and meets all quality requirements.

Reusable Components -

Domain model, software architecture, product design, and implementation components (source code, test plans, procedures and results, and system/software and process documentation).

Reuse Library -

A library specifically designed, built, and maintained to house reusable components.

Small Scale Reuse -

Small scale reuse is the reapplication of code: subroutines, object libraries, or Ada packages.

Software Architecture -

A specification for the assemblage of components.

Systematic Reuse -

Reuse is practiced systematically when there exist defined procedures for leveraging future software projects. Efforts are devoted up-front to creating a suitable process.

Vertical Domain -

The knowledge and concepts that pertain to a particular application domain.

Wrapper -

A component which allows integration of components. A wrapper is used when some capabilities required of the component are not provided by the product or when the product's interface does not completely match that of the component.

A.10 HANDOUT - ACRONYMS

AJPO	Ada Joint Program Office
ASSET	Asset Source for Software Engineering Technology
BS	Bachelor of Science
CAMP	Common Ada Missile Packages
CARDS	Central Archive for Reusable Defense Software
CIM	Center for Information Management
COTS	Commercial-Off-The-Shelf
DISA	Defense Information Systems Agency
DoD	Department of Defense
DSRS	Defense Software Repository System
DSSA	Domain Specific Software Architecture
FODA	Feature-Oriented Domain Analysis
GOTS	Government-Off-The-Shelf

MS	Master of Science
NEC	Nippon Electronics Corporation
NTTC	National Technology Transfer Center
PRISM	Portable, Reusable, Integrated Software Modules
RAPID	Reusable Ada Packages for Information System Development
RFP	Request For Proposal
RIG	Reuse library Interoperability Group
RLF	Reusability Library Framework
RNTDS	Restructured Naval Tactical Data Systems
SDESS	Software Development for Electronic Switching Systems
STARS	Software Technology for Adaptable, Reliable Systems

A.11 HANDOUT - BIBLIOGRAPHY

[ANTHES92]	U.S. Software Reuse Plan Draws Criticism, Computer World, G. Anthes, Aug 92.
[ASSET91]	ASSET Set-Up Plans, Software Technology for Adaptable, Reliable, Systems (STARS) Program, 9 Mar 91.
[ASSET92]	Criteria and Implementation Procedures for Evaluation of Reusable Software Engineering Assets, ASSET, The National Software Technology Repository, IBM, Mar 92.
[BIGG89]	Software Reusability, Volume 1, Concepts and Models, ACM Press, T. J. Biggerstaff, A. J. Perlis, 89.
[BOEHM81]	Software Engineering Economics, Prentice Hall, B. Boehm, 81.
[BOEHM92]	Megaprogramming (preliminary version), STARS 92, On The Road to Megaprogramming, SEE, Vol. 3, DARPA, B. Boehm, W. Scherlis, 92.
[CARDS90]	ProtoTech: Process Synergy With STARS, STARS 92, On The Road to Megaprogramming, Vol. 1, W. Carlson, 8 Dec 92.

[CARDS92a]	Direction Level Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04104/001/00, 20 Nov 92.
[CARDS92b]	Library Operations Policies and Procedures, Central Archive for Reusable Defense Software (CARDS), STARS-AC-4109/001/00, 25 Nov 92.
[CARDS92c]	Command Center Domain Model Description, Central Archive for Reusable Defense Software (CARDS), Draft - STARS-AC-04110/001/00, 25 November 92.
[CARDS92d]	Library Development Handbook (LDH), Central Archive for Reusable Defense Software (CARDS), STARS-AC-04109/001/00, 25 November 92.
[CARDS93a]	Engineer's Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-VC-B008/001/00, 30 September 93.
[CARDS93b]	Component and Tool Developer's Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04114/001/00, 15 Mar 93.
[CARDS93c]	Acquisition Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04105/001/00, 30 Apr 93.
[CARDS94a]	Technical Concept Document, Central Archive for Reusable Defense Software (CARDS), STARS-VC-B009/001/00, 28 Feb 94.
[CARDS94b]	Training Plan, Central Archive for Reusable Defense Software (CARDS), STARS-VC-B003/001/00, 29 Jan 94.
[COHEN91]	Application of Feature Oriented Domain Analysis to Army Movement Control Domain, Software Engineering Institute, (SEI), CMU/SEI-91-TR-28, S. Cohen, J. Stanley, A. Peterson, R. Krut, 30 Sep 91.
[COHEN92]	Modeling Software Reuse Technology: Feature Oriented Domain Analysis (FODA) - Tutorial Slides, SEI, Carnegie Mellon University, May 92.
[DISA92a]	Software Reuse Initiative Program Overview, DoD Reuse Initiative, Defense Information Systems Agency

	(DISA)/Center for Information Management (CIM), Presented by SofTech, Inc., 2 Nov 92.
[DISA92b]	DoD Domain Analysis Guidelines, DoD Software Reuse Initiative, Defense Information Systems Agency (DISA)/Center for Information Management (CIM), May 92.
[DISA92c]	Department of Defense, Center for Software Reuse Operations, Training Plan, 27 Jul 92.
[DoD92]	DoD Software Reuse Vision and Strategy, CrossTalk, Oct 92.
[FRAKES90]	Representing Reusable Software, Information and Software Technology, Vol. 32, No. 10, W. B. Frakes, P. B. Gandel, 10 Dec 90.
[FRAKES92]	Software Reuse, Domain Analysis, and Reengineering, W. Frakes, R. Prieto-Diaz, R. Arnold, 92.
[GOOD92]	Restructured Naval Tactical Data System (RNTDS) An Example of Applying Megaprogramming Concepts, Stars 92, T. J. Goodall, 8 Dec 92.
[HESS90]	A Domain Analysis Bibliography, Software Engineering Institute (SEI), CMU/SEI-90-SR-3, J. Hess, W. Novak, P. Carroll, S. Cohen, R. Holibaugh, K. Kang, A. Peterson, Jun 90.
[HISS92]	Domain-Specific Reuse for Post-Deployment Maintenance, Central Archive for Reusable Defense Software (CARDS), S. Hissam, 92.
[HOLI89]	Joint Integrated Avionics Working Group (JIAWG) Domain Analysis Concepts, R. Holibaugh, 19 Dec 89.
[HOOP89]	Software Reuse Guidelines, U.S. Army Institute for Research, J. W. Hooper, R. O. Chester, ASQB-GI-90-015, Dec 89.
[MITRE92]	A New Process for Acquiring Software Architecture, The MITRE Corporation, T. F. Saunders, Dr. B. M. Horowitz, M. L. Mleziva, M920000126, Nov 92.
[NATO]	Software Reuse Procedures, NATO Communications and Information Agency, Vol. 3 of 3.

[PAYTON92]	Domain-Specific Reuse, Context, Accomplishments and Directions, STARS 92, On the Road to Megaprogramming, Vol. 2, 8 Dec 92.
[PRES87]	Software Engineering A Practitioner's Approach, Second Edition, R. Pressman, McGraw-Hill Book Company, 87.
[PRIETO90]	Domain Analysis: An Introduction, ACM SigSoft, Software Engineering Notes, R. Prieto-Diaz, Vol. 15, No. 2, Apr 90.
[PRIETO91a]	Making Software Reuse Work: An Implementation Model, First International Workshop on Software Reusability, R. Prieto-Diaz, 5-6 Jul 91.
[PRIETO91b]	Implementing Faceted Classification for Software Reuse, Communications of the ACM, Vol. 34, No. 5, R. Prieto-Diaz, May 91.
[PRISM92]	Qualification Methodology Report, Portable Reusable, Integrated Software Modules (PRISM) Program, 14 Jul 92.
[RAPID]	The RAPID Center Reusable Software Components (RSCs) Certification Process, U.S. Army Information Systems Software Development Center, Washington, J. Piper and W. Barner.
[RAPID91]	RAPID Qualifying Software Components: Reusability Metrics, Software Reuse and Re-Engineering Conference for the National Institute for Scftware Quality and Productivity, A. Nieder, 30 Apr 91.
[SEI92]	A Reuse-Based Software Development Methodology, CMU/SEU-92-SR-4, K. Kang, S. Cohen, R. Holibaugh, J. Perry, A. Peterson, Jan 92.
[SPC92]	Reuse Adoption Guidebook, Software Productivity Consortium, SPC-92051-CMC, Version 01.00.03, Nov 92.
[STARS91a]	US40 - Domain-Specific Environment Repository Composite Paradigm Report, STARS-SC-03068/001/00, 30 May 91.
[STARS91b]	US40 - Risk-Reduction Reasoning-Based Development Paradigm Tailored to Navy C2 Systems, STARS-SC-03070/001/00, 30 Jul 91.

[STARS92a]	STARS Reuse Concepts, Volume 1, Conceptual Framework for Reuse Processes (CFRP), Version 2, STARS-UC-05159/001/00, 13 Nov 92.
[STARS92b]	Abridged AdaKNET User's Manual, Software Technology for Adaptable, Reliable Systems (STARS), Draft Version 3.1, 6 May 92.
[STARS92c]	STARS Reuse Concepts, Volume 1, Conceptual Framework for Reuse Processes (CFRP), Version 1, STARS-TC-04040/001/00, 14 Feb 92.
[TOMA91]	Software Engineering Education, SEI Conference 1991, J. Tomayko, Springer-Verlag, Oct 91.
[TRACZ92]	Domain-Specific Software Architecture Engineering Process Guidelines, ADAGE-IBM-92-02, Version 1.0, 17 Mar 92.
[WAL92]	CARDS: A Blueprint and Environment for Domain-Specific Software Reuse, Central Archive for Reusable Defense Software (CARDS), STARS 92, On The Road to Megaprogramming, Vol. 2, K. Wallnau, 8 Dec 92.
[WART92]	Criteria for Comparing Reuse-Oriented Domain Analysis Approaches, Software Productivity Consortium, S. Wartik, R. Prieto-Diaz, 1991.
[YOUR92]	Decline and Fall of the American Programmer, Yourdon Press, E. Yourdon, 92.

APPENDIX B - Introduction to Reuse

B.1 INTRODUCTION

B.1.1 PURPOSE

This course has been developed to provide an introduction to the concepts associated with software reuse. It explains what software reuse is, the processes used to accomplish reuse, and potential benefits and risks. This course is intended for use in Government and industry training. It can be tailored for presentation at the university level.

B.1.2 CARDS PROGRAM MISSION

The Central Archive for Reusable Defense Software (CARDS) Program is a concerted Department of Defense (DoD) effort to transition advances in the techniques and technology of library-centered, domain-specific software reuse into mainstream DoD software procurements. There are three key elements to the CARDS approach:

1. Apply domain-specific reuse techniques and technology to produce an operational library for command centers

2. Develop and transition, through a Franchise Plan, the "knowledge" for domain-specific reuse to the DoD and DoD Software Development Industry

3. Develop and transition a training plan and training courses as a vehicle for enhancing the acceptance of the Franchise Plan.

4. The domain-specific reuse knowledge gained during the CARDS effort will be conveyed via a Franchise Plan and three sets of documents: Reuse Adoption Handbooks, CARDS library operation and maintenance related documents, and training and educational material.

B.1.3 RELATIONSHIP TO OTHER CARDS DOCUMENTS

The Franchise Plan provides a description of reuse processes and instructions for tailoring development processes to effect domain-specific reuse. It describes, in precise steps, a scenario for an organization to establish a domain-specific reuse capability.

The Reuse Adoption Handbooks consist of the Component and Tool Developer's, Acquisition, Direction Level, and Engineer's Handbooks. Together these four handbooks address software development, program management, and executive planning. The Component and Tool Developer's Handbook addresses the development of reusable software components, emphasizing evaluation and acceptance criteria required by libraries and users of the components [CARDS93b]. The Acquisition Handbook assists Government Program Managers and their support staff in incorporating software reuse into the acquisition and maintenance portions of the life-cycle process. The Acquisition Handbook provides guidance in planning the acquisition strategy, contract award, managing the effort, and follow-on support [CARDS93c]. The Direction Level Handbook offers a framework to assist Government acquisition executives in establishing plans to manage software reuse across their systems. Importance is placed on the policy and business issues (e.g., regulations, incentives, funding, cost/benefit, education and training, and ownership of components) that act as the support structure for reuse [CARDS92a]. The Engineer's Handbook assists software engineers and other technical personnel in integrating software reuse development methods and techniques into their own software engineering processes. The focus of the Engineer's Handbook is on how reuse impacts each activity of the software development process, from requirements analysis through maintenance [CARDS93a].

Although some of the CARDS library operation and maintenance documents are specific to the CARDS library, they can be used by other organizations to learn how reuse was implemented in the command and control domain. These CARDS documents address the library's operations procedures, the technical concepts, project management plans, as well as describing the domain model.

The CARDS training effort includes a training plan, course outlines, and sample course materials relating to topics included in each Reuse Adoption Handbook. They are geared to educate the software professional and support the reduction of cultural barriers to reuse. They can be tailored to meet the needs of varying audiences. This course has been developed as a means of introducing the concepts of software reuse.

B.1.4 DOCUMENT ORGANIZATION

The course description begins by providing the reader with a description of the intended course audience. Section 2 characterizes the student. Section 3 provides the reader with a guideline for selecting an instructor to present the course. Section 4 provides a course overview. Course content for each of the units is described in detail in Section 5.

Included with this course description is a sample set of course materials. The materials can be used as they are, or modified to suit the needs of the presenting organization.

B.2 STUDENT

B.2.1 STUDENT CHARACTERIZATION

This course is intended for experienced system and software engineers.

B.2.1.1 System Engineers

System engineers are concerned with the decomposition of systems, the allocation of software development responsibility for specific system components to software engineers, and with the subsequent composition of software and hardware system components to produce the final system [TOMA91].

System engineers begin with customer-defined goals, requirements and constraints and derive a representation of function, performance, interfaces design constraints, and information structure that can be allocated to each of the generic system elements [PRES87].

B.2.1.2 Software Engineer

To accommodate function and performance, defined during system engineering, software engineers must build or acquire a set of software components [PRES87]. The software engineer is responsible for analyzing software requirements, developing or identifying existing software designs and software components, and integrating, testing, and maintaining software components.

B.2.2 STUDENT PREREQUISITE KNOWLEDGE

Successful completion of this course requires that the student possess prior experience participating in a complex software development project (e.g., at least 100K Source Lines of Code).

B.3 INSTRUCTOR

B.3.1 JOB DESCRIPTION

The instructor is responsible for conducting a tailored implementation of the recommended course content through a lecture/activity format. This is accomplished by completing the following tasks:

1. Consulting the reference documents,
2. Reviewing the recommended outline of course content and choosing appropriate levels of content based on audience and time,
3. Preparing appropriate training materials,
4. Setting up activities,
5. Ensuring the atmosphere of the training session is informal, relaxing, intellectually stimulating, and student-centered,
6. Integrating lecture and discussion into the training session,
7. Conducting the training session.

B.3.2 FORMAL EDUCATION

The instructor should hold at least a BS degree in software/system engineering (an MS degree is preferred) or a degree in computer science or a closely related field.

B.3.3 KNOWLEDGE OF INSTRUCTION

The instructor should possess an understanding of the principles of learning and teaching methodologies, an ability to apply these principles and methods, and excellent communication skills.

B.3.4 PRACTICAL TEACHING EXPERIENCE

The instructor should have at least two years experience teaching short courses, and should possess experience addressing system and software engineering personnel.

B.3.5 KNOWLEDGE OF SUBJECT

The instructor must be knowledgeable in software reuse and understand the impact of integrating software reuse into system and software engineering processes. The instructor must have completed formal training in the form of a workshop or seminar on domain-specific software reuse. For presentation of the course to DoD personnel, the instructor should also have knowledge of general DoD engineering and procurement standards as well as the various DoD mandates effecting system development and maintenance.

B.3.6 SKILL IN PERFORMANCE

The instructor should be able to demonstrate the ability to perform the tasks required in the practical application of the training course.

B.4 COURSE OVERVIEW

B.4.1 COURSE BREAKDOWN

The content for this course is broken into five units. These units can be presented in dependently or contiguously. Students should attend units in the order described; however, units can be skipped if students possess the knowledge to be presented. The units comprising this course are as follows:

Unit 1: Introduction and Rationale Anticipated Duration: 2 hours

An introduction to the DoD Software Reuse Vision and Strategy [DOD92] is provided. An overview of the CARDS Program is presented and DoD reuse coordination efforts are discussed.

If the presenting organization is not CARDS, an appropriate introduction should replace this unit. It is recommended that the introduction include an overview of the presenting organization and its rationale for teaching domain-specific reuse.

Unit 2: Software Reuse Concepts Anticipated Duration: 2 hours

An overview of software reuse is presented. This unit addresses why reuse should be considered and how long reuse has been practiced. Reusable components are defined. Differences in reuse approaches are discussed (e.g., ad hoc, opportunistic, systematic, large scale reuse, small scale reuse). Finally, potential risks and benefits associated with reuse are presented.

Unit 3: Domain-Specific Reuse Anticipated Duration: 2 hours

A brief history of domain-specific reuse begins this unit. An overview of domain engineering is provided and the products of domain engineering are discussed. Application engineering within the domain-specific reuse framework is introduced and explored. Students learn how domain engineering products are incorporated into every software development activity from requirements analysis through maintenance.

Unit 4: Software Reuse Libraries Anticipated Duration: 2 hours

An overview of component-based and model-based libraries is provided. Students are introduced to various types of library representations and mechanisms, and certification and qualification of library components. The intent of this unit is to build trust in the students for reusable components.

B.5 ANNOTATED COURSE OUTLINE

B.5.1 UNIT 1 - INTRODUCTION AND RATIONALE

B.5.1.1 Learning Objectives

This introduction has been developed for presentation of the course by CARDS. Since CARDS is a concerted DoD effort, it is necessary for students to understand the software reuse goals of the DoD. The objectives of this lecture are to provide students with an understanding of the DoD Software Reuse Vision and Strategy document, the CARDS Program goals and products, and the role of CARDS in the DoD's current reuse efforts. If this course is being presented by an organization other than CARDS, this unit can be replaced in its entirety, or the CARDS overview may be removed and replaced with an overview of the presenting organization.

B.5.1.2 Lecture Content Area

A. DoD Software Reuse Vision and Strategy Overview

The DoD Software Reuse Vision and Strategy [DOD92] describes an initiative which will make a reuse-based paradigm the preferred alternative for developing and supporting software. Students will learn what this document is and how it may effect the way they currently practice software development.

B. CARDS Overview [WAL92]

The Central Archive for Reusable Defense Software (CARDS) Program is a concerted Department of Defense (DoD) effort to transition advances in the techniques and technology of architecture-based, library-assisted, domain-specific software reuse into mainstream DoD software procurements. Students will learn what the goals of the CARDS Program are, what activities CARDS is involved in, and what products and services CARDS has to offer the DoD soft ware community. This is intended to be a brief overview. Students will be directed to specific documents for additional information.

C. Interoperability Coordination Among DoD Reuse Efforts

CARDS is not a program in isolation, but one of several related reuse initiatives. Students will be provided with an overview of several of the major DoD domain-specific operational reuse library efforts and how they are coordinating to achieve interoperability. Students will learn about the Reuse Library Interoperability Group (RIG) and why CARDS is a participating member. Students will gain an understanding of how CARDS, the STARS Asset Source for Software Engineering Technology (ASSET), and the Defense Information Systems Agency (DISA) Center for Information Management (CIM) Defense Software Repository System (DSRS) are coordinating.

B.5.2 UNIT 2 - SOFTWARE REUSE CONCEPTS

B.5.2.1 Learning Objectives

The objectives of this lecture are to provide the students with an understanding of software reuse and an appreciation of the potential benefits of reuse.

B.5.2.2 Lecture Content Area

Software reuse is the reapplication of domain knowledge, development experience, design decisions, architectural structures, requirements, designs, code, and documentation from existing systems to an emerging system in an effort to reduce the costs associated with software development and maintenance [BIGG89]. Cost includes manpower (skills and availability), risks to the schedule, and software and hardware budgets.

B.5.3 UNIT 3 - DOMAIN-SPECIFIC REUSE

B.5.3.1 Learning Objectives

The objectives of this lecture are to provide the students with an understanding of domain-specific reuse. Students will learn about domain engineering, the activities it encompasses, and the products that result. They will learn how the process of application engineering is performed using the products of domain engineering.

B.5.3.2 Lecture Content Area

Domain-specific reuse, which utilizes a generic architecture, is a key aspect of the DoD Software Reuse Vision and Strategy. It is believed that domain-specific reuse can result in greater savings than general-purpose reuse [DOD92]. A domain is an area of activity or knowledge containing applications which share a set of common capabilities and data. Domains can be defined as vertical or horizontal. A vertical domain is a specific class of system, such as command and control or weapon systems. A horizontal domain consists of general software functions that are applicable across multiple vertical domains, such as user interfaces, mathematical programs, and graphics packages [CARDS93c]. As a domain matures, so matures the body of knowledge about it and experience in it. A mature domain has a larger number of existing systems and domain experts from which information can be drawn. Components which have been maintained and refined as the domain matures will become more reliable and effective.

B.5.4 UNIT 4 - SOFTWARE REUSE LIBRARIES

B.5.4.1 Learning Objectives

The objectives of this lecture are to introduce the students to reuse libraries, library mechanisms, and certification and qualification techniques. Students will receive hands-on experience using a reuse library.

B.5.4.2 Lecture Content Area

A. The Library - A Reuse Tool

Software reuse libraries are a tool for reuse; they are not the reuse solution. The instructor will discuss how most individuals maintain their own pool of reusable components and how compilers are equipped with their own set of reusable libraries. The students will gain an understanding of how the software reuse library concept is now being extended to include publicly accessible libraries of life-cycle artifacts. This extension is essential if the full potential of software reuse is to be realized. The greatest reuse savings can be seen when a component is reused many times, across projects. organizations may benefit by making their own components available internally to all projects and by acquiring components available externally [NATO].

B. Types of Libraries

Reuse libraries can be classified by the methods and techniques used to organize components. A library is either component-based or model-based [WAL92]. Students will learn how libraries of these two classes are similar and how they differ.

B.6 HANDOUT - COURSE EVALUATION FORM

Please complete the following course evaluation form.

1. Name (Optional):
2. Organization (Optional):
3. I perform the following tasks as part of my job responsibilities (circle all that apply):

   Specify systems/software

   Design systems/software

   Develop systems/software

   Test systems/software

   Verify systems/software

   Control systems/software

   Manage systems/software

   Other___________
4. I will be able to apply the course content to my job duties (Y/N).
5. Before I attended this course, I was familiar with:

Table 5-1

	Novice	Knowledgable	Experienced	Expert
Structured Analysis				
Model-Based Engineering				
Domain Analysis				
Software Reuse Libraries				
Domain-Specific Software Reuse Libraries				
Generic Architecture Development				

System Engineering				
Software Engineering				

6. The length of this class was:

   Too Long

   Too Short

   Just Right

7. The material covered was:

   Too Specific

   Too General

   Adequate

8. The time given for each unit was adequate: (Y/N)

   If no, indicate which units did not provide adequate time.

9. *I had plenty of opportunity to participate and ask questions:* (Y/N)

10. Were your questions adequately answered? (Y/N)

11. The examples helped clarify the concepts:

    Very Much

    Moderately

    Not at All

12. The course handouts were:

    Very Useful

    Moderately Useful

    Not Useful

13. I would recommend attendance to my colleagues: (Y/N)

    If no, why not?

14. I have attended other software reuse training classes or seminars: (Y/N)

    If yes, which ones?

15. I would be interested in attending other software reuse training seminars: (Y/N)

16. I would be interested in receiving additional information on software reuse: (Y/N)

17. I believe domain-specific software reuse is very valuable in the development of software systems: (Y/N)

18. The training facilities and equipment were adequate: (Y/N)

19. The instructor's knowledge of course material was:

    Poor

    Fair

    Good

    Excellent

20. The instructor's ability to explain the course material was:

    Poor

    Fair

    Good

    Excellent

21. The instructor's preparation of course material was:

    Poor

    Fair

    Good

    Excellent

22. Additional Comments:

B.7 HANDOUT - GLOSSARY

Ad-Hoc Reuse -

Reuse is practiced ad-hoc when there are no defined methods for performing reuse.

Application -

A system which provides a set of general services for solving some type of user problem.

Architectural Constraints -

The definition of relationships/interfaces among components of a generic architecture.

Architecture-Centric Reuse -

Reuse is architecture-centric when the component development process and the application development processes are based on a generic architecture. The goal of an architecture-driven process is to achieve black-box reuse.

Black-box Reuse -

Black-box reuse is achieved when application engineers can compose systems by plugging together different reusable components based on an application's requirements.

Certification -

The process of determining that a component being considered for inclusion in the library meets the requirements of the library and passes all testing procedures.

Common Metrics -

Criteria used to evaluate components regardless of domain. Example criteria for common metrics are reliability, maintainability, portability, and security.

Component-Based Library -

Component-based libraries are similar to book libraries. They can be thought of as software warehouses. The central focus of a component-based library is the component.

Domain -

The functional area covered by a family of systems (e.g., the aircraft navigation systemdomain) or across systems where similar software requirements exist.

Domain Analysis -

The process of identifying, collecting, organizing, analyzing, and representing the relevant information in a domain based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain.

Domain Engineering -

An encompassing process which includes domain analysis and the subsequent construction of components, methods, tools, and supporting documentation that address the problems of system/subsystem development through the application of the knowledge in the domain model and software architecture.

Domain Metrics -

Criteria used to evaluate components based on domain requirements, architectural constraints, and implementation constraints.

Domain Model -

A definition of the functions, objects, data, and relationships in a domain, consisting of a concise representation of the commonalities and differences of the problems of the domain and their solutions.

Domain-Specific Reuse -

Reuse that is targeted for a specific domain (as opposed to reuse of general purpose work products). It typically involves reuse of larger workproducts (subsystems, architectures, etc.) than general purpose reuse.

Domain-Specific Software Architecture -

A software architecture that has been generalized based on actual and projected commonalities of systems in a domain. May also be referred to as a generic architecture.

Facet -

A perspective, viewpoint, or dimension of a particular domain which can be represented as a class.

Generic Architecture -

High-level paradigms and constraints that characterize the commonality and variances of the interactions and relationships between the various components in a system. May also be referred to as a domain specific software architecture.

Horizontal Domain -

The knowledge and concepts that pertain to a particular functionality of a set of software components that can be utilized across more than one application.

Implementation Constraints -

Constraints to be satisfied when components of a generic architecture are composed into specific systems.

Interoperability -

The ability of two or more systems to exchange information and to mutually use the information that has been exchanged.

Knowledge Blueprint -

A flexible plan to transition knowledge to the community.

Large Scale Reuse -

Large scale reuse is the reapplication of high-level components (e.g., requirements, architectures, designs).

Library-Assisted Reuse -

Reuse is library-assisted when there exists a library to support the application domain. There may be more than one library and they may be interconnected.

Megaprogramming -

Megaprogramming is achieved when systems and subsystems can be viewed as black-boxes that meet certain requirements. These systems can be reused in building other systems without the developer requiring detailed knowledge of the system's internal structures.

Model-Based Library -

Model-based libraries are organized around the principle that what matters in a repository is the context in which reusable software components are used and the relationships among components. The focus of a model-based library is the model (requirements, architectures, design decisions and rationales) and the software which implements these models.

Opportunistic Reuse -

Reuse is practiced opportunistically when it is up to software developers to identify when reuse is possible, locate reusable components, and integrate them.

Process-Driven Reuse-

Software reuse is process-driven when it is an integral and transparent part of both the software engineering process and the broader acquisition process.

Prototype -

A model of a system (sometimes scaled down, but accurate) used to verify the operational process prior to building a final system.

Qualification -

The process of determining that a potential component is appropriate to the library and meets all quality requirements.

Reusable Components -

A representation of some aspect of a system which may be used in different applications. A component may consist of requirements, architectures, designs or implementation (e.g., code, tests, documentation) information.

Reuse Library -

A library specifically designed, built, and maintained to house reusable components.

Small Scale Reuse -

Small scale reuse is the reapplication of code: subroutines, object libraries, or Ada packages.

Software Architecture -

A specification for the assemblage of components.

Systematic Reuse -

Reuse is practiced systematically when there exist defined procedures for leveraging future software projects. Efforts are devoted up-front to creating a suitable process.

Vertical Domain -

The knowledge and concepts that pertain to a particular application domain.

Wrapper -

A component which allows integration of components. A wrapper is used when some capabilities required of the component are not provided by the product or when the product's interface does not completely match that of the component.

B.8 HANDOUT - ACRONYMS

AJPO	Ada Joint Program Office
ASSET	Asset Source for Software Engineering Technology
BS	Bachelor of Science
CAMP	Common Ada Missile Packages

CARDS	Central Archive for Reusable Defense Software
CIM	Center for Information Management
COTS	Commercial-Off-The-Shelf
DISA	Defense Information Systems Agency
DoD	Department of Defense
DSRS	Defense Software Repository System
DSSA	Domain Specific Software Architecture
FODA	Feature-Oriented Domain Analysis
GOTS	Government-Off-The-Shelf
MS	Master of Science
NEC	Nippon Electronics Corporation
NTTC	National Technology Transfer Center
PRISM	Portable, Reusable, Integrated Software Modules
RAPID	Reusable Ada Packages for Information System Development
RFP	Request For Proposal
RIG	Reuse library Interoperability Group
RLF	Reusability Library Framework
RNTDS	Restructured Naval Tactical Data Systems
SDESS	Software Development for Electronic Switching Systems
STARS	Software Technology for Adaptable, Reliable Systems

B.9 HANDOUT - BIBLIOGRAPHY

[ANTHES92]	U.S. Software Reuse Plan Draws Criticism, Computer World, G. Anthes, Aug 92.
[ARMY92]	ARMY Reuse Plan, 31 Aug 92.
[ASSET91]	ASSET Set-Up Plans, Software Technology for Adaptable, Reliable, Systems (STARS) Program, 9 Mar 91.

[ASSET92]	Criteria and Implementation Procedures for Evaluation of Reusable Software Engineering Assets, ASSET, The National Software Technology Repository, IBM, Mar 92.
[BIGG89]	Software Reusability, Volume 1, Concepts and Models, ACM Press, T. J. Biggerstaff, A. J. Perlis, 89.
[BOEHM81]	Software Engineering Economics, Prentice Hall, B. Boehm, 81.
[BOEHM92]	Megaprogramming (preliminary version), STARS 92, On The Road to Megaprogramming, SEE, Vol. 3, DARPA, B. Boehm, W. Scherlis, 92.
[CARDS92a]	Direction Level Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04104/001/00, 20 Nov 92.
[CARDS92b]	Library Operations Policies and Procedures, Central Archive for Reusable Defense Software (CARDS), STARS-AC-4109/001/00, 25 Nov 92.
[CARDS92c]	Command Center Domain Model Description, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04110/001/00, 25 November 92.
[CARDS92d]	Library Development Handbook (LDH), Central Archive for Reusable Defense Software (CARDS), STARS-AC-04109/001/00, 25 November 92.
[CARDS93a]	Engineer's Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-VC-B008/001/00, 30 September 93.
[CARDS93b]	Component and Tool Developer's Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04114/001/00, 15 Mar 93.
[CARDS93c]	Acquisition Handbook, Central Archive for Reusable Defense Software (CARDS), STARS-AC-04105/001/00, 30 Apr 93.
[CARDS94a]	Technical Concept Document, Central Archive for Reusable Defense Software (CARDS), STARS-VC-B009/001/00, 28 Feb 94.

[CARDS94b]	Training Plan, Central Archive for Reusable Defense Software (CARDS), STARS-VC-B003/001/00, 29 Jan 94.
[COHEN92]	Modeling Software Reuse Technology: Feature Oriented Domain Analysis (FODA) - Tutorial Slides, SEI, Carnegie Mellon University, May 92.
[DISA92a]	Software Reuse Initiative Program Overview, DoD Reuse Initiative, Defense Information Systems Agency (DISA)/Center for Information Management (CIM), Presented by SofTech, Inc., 2 Nov 92.
[DISA92b]	DoD Domain Analysis Guidelines, DoD Software Reuse Initiative, Defense Information Systems Agency (DISA)/Center for Information Management (CIM), May 92.
[DISA92c]	Department of Defense, Center for Software Reuse Operations, Training Plan, 27 Jul 92.
[DoD92]	DoD Software Reuse Vision and Strategy, CrossTalk, Oct 92.
[FRAKES90]	Representing Reusable Software, Information and Software Technology, Vol. 32, No. 10, W. B. Frakes, P. B. Gandel, 10 Dec 90.
[FRAKES92]	Software Reuse, Domain Analysis, and Reengineering, W. Frakes, R. Prieto-Diaz, R. Arnold, 92.
[GOOD92]	Restructured Naval Tactical Data System (RNTDS) An Example of Applying Megaprogramming Concepts, Stars 92, T. J. Goodall, 8 Dec 92.
[HAYH92]	Component Identification & Qualification, Central Archive for Reusable Defense Software (CARDS), B. Hayhurst, B. Curfman, 21 Oct 92.
[HESS92]	Army Software Reuse Plan, Status Review, J. Hess, 5 Aug 92.
[HISS92]	Domain-Specific Reuse for Post-Deployment Maintenance, Central Archive for Reusable Defense Software (CARDS), S. Hissam, 92.
[HOOP89]	Software Reuse Guidelines, U.S. Army Institute for Research, J. W. Hooper, R. O. Chester, ASQB-GI-90-015, Dec 89.

[LEVIN92]	Reusable Software Components, CrossTalk, T. Levin, Mar 92.
[MITRE92]	A New Process for Acquiring Software Architecture, The MITRE Corporation, T. F. Saunders, Dr. B. M. Horowitz, M. L. Mleziva, M920000126, Nov 92.
[NATO]	Software Reuse Procedures, NATO Communications and Information Agency, Vol. 3 of 3.
[PAYTON92]	Domain-Specific Reuse, Context, Accomplishments and Directions, STARS 92, On the Road to Megaprogramming, Vol. 2, 8 Dec 92.
[PRIETO90]	Domain Analysis: An Introduction, ACM SigSoft, Software Engineering Notes, R. Prieto-Diaz, Vol. 15, No. 2. Apr 90.
[PRIETO91a]	Making Software Reuse Work: An Implementation Model, First International Workshop on Software Reusability, R. Prieto-Diaz, 5-6 Jul 91.
[PRIETO91b]	Implementing Faceted Classification for Software Reuse, Communications of the ACM, Vol. 34, No. 5, R. Prieto-Diaz, May 91.
[PRISM92]	Qualification Methodology Report, Portable Reusable, Integrated Software Modules (PRISM) Program, 14 Jul 92.
[RAPID91]	RAPID Qualifying Software Components: Reusability Metrics, Software Reuse and Re-Engineering Conference for the National Institute for Software Quality and Productivity, A. Nieder, 30 Apr 91.
[SPC92]	Reuse Adoption Guidebook, Software Productivity Consortium, SPC-92051-CMC, Version 01.00.03, Nov 92.
[STARS92a]	STARS Reuse Concepts, Volume 1, Conceptual Framework for Reuse Processes (CFRP), Version 2, STARS-UC-05159/001/00, 13 Nov 92.
[STARS92b]	Abridged AdaKNET User's Manual, Software Technology for Adaptable, Reliable Systems (STARS), Draft Version 3.1, 6 May 92.
[STARS92c]	STARS Reuse Concepts, Volume 1, Conceptual Framework for Reuse Processes (CFRP), Version 1, STARS-TC-04040/001/00, 14 Feb 92.

[STARS92d]	RLF Graphical Browser User's Manual, Software Technology for Adaptable, Reliable Systems (STARS), STARS-TC-04046/005/00, 1 Jul 92.
[TRACZ92]	Domain-Specific Software Architecture Engineering Process Guidelines, ADAGE-IBM-92-02, Version 1.0, 17 Mar 92.
[WAL92]	CARDS: A Blueprint and Environment for Domain-Specific Software Reuse, Central Archive for Reusable Defense Software (CARDS), STARS 92, On The Road to Megaprogramming, Vol. 2, K. Wallnau, 8 Dec 92.
[WART92]	Criteria for Comparing Reuse-Oriented Domain Analysis Approaches, Software Productivity Consortium, S. Wartik, R. Prieto-Diaz, 1991.
[YOUR92]	Decline and Fall of the American Programmer, Yourdon Press, E. Yourdon, 92.